Artificial Intelligence 226 (2015) 102–121

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Transferring knowledge as heuristics in reinforcement learning: A case-based approach

Reinaldo A.C. Bianchi [a,*], Luiz A. Celiberto Jr. [b,1], Paulo E. Santos [a,2], Jackson P. Matsuura [c,3], Ramon Lopez de Mantaras [d,4]

[a] *Centro Universitário da FEI, Av. Humberto de A.C. Branco, 3972, São Bernardo do Campo, São Paulo, Cep: 09850-901, Brazil*
[b] *Universidade Federal do ABC (UFABC), Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas – CECS, Avenida dos Estados, 5001, Santo André, São Paulo, Cep: 09210-580, Brazil*
[c] *Technological Institute of Aeronautics (ITA), Praça Marechal Eduardo Gomes, 50, S.J.C., São Paulo, Cep: 12.228-900, Brazil*
[d] *IIIA – Artificial Intelligence Research Institute, CSIC – Spanish National Research Council, Campus Universitat Autonoma de Barcelona, 08193 Bellaterra, Catalonia, Spain*

ARTICLE INFO

Article history:
Received 10 December 2013
Received in revised form 9 December 2014
Accepted 24 May 2015
Available online 29 May 2015

Keywords:
Case-based reasoning
Reinforcement learning
Transfer learning

ABSTRACT

The goal of this paper is to propose and analyse a transfer learning meta-algorithm that allows the implementation of distinct methods using heuristics to accelerate a Reinforcement Learning procedure in one domain (the target) that are obtained from another (simpler) domain (the source domain). This meta-algorithm works in three stages: first, it uses a Reinforcement Learning step to learn a task on the source domain, storing the knowledge thus obtained in a case base; second, it does an unsupervised mapping of the source-domain actions to the target-domain actions; and, third, the case base obtained in the first stage is used as heuristics to speed up the learning process in the target domain. A set of empirical evaluations were conducted in two target domains: the 3D mountain car (using a learned case base from a 2D simulation) and stability learning for a humanoid robot in the Robocup 3D Soccer Simulator (that uses knowledge learned from the Acrobot domain). The results attest that our transfer learning algorithm outperforms recent heuristically-accelerated reinforcement learning and transfer learning algorithms.

1. Introduction

Reinforcement learning (RL) is a field of machine learning whose aim is to maximise the total amount of reward an agent receives while interacting with its environment [1]. This interaction occurs by means of exploring the state space by trial-and-error actions on the environment, leading to a process whose convergence is often slow (or infeasible) on complex tasks [2,3].

* Corresponding author. Tel.: +55 11 4353 2900.
E-mail addresses: rbianchi@fei.edu.br (R.A.C. Bianchi), luiz.celiberto@ufabc.edu.br (L.A. Celiberto), psantos@fei.edu.br (P.E. Santos), jackson@ita.br (J.P. Matsuura), mantaras@iiia.csic.es (R. Lopez de Mantaras).
[1] Tel.: +55 11 4996 0123.
[2] Tel.: +55 11 4353 2900.
[3] Tel.: +55 11 4353 2900.
[4] Tel.: +34 93 580 9570.

http://dx.doi.org/10.1016/j.artint.2015.05.008
0004-3702/© 2015 Elsevier B.V. All rights reserved.

The current scientific literature presents distinct ways of accelerating the computational process involved in Reinforcement Learning (RL) through the use of various methods, such as composition of functions [4], human feedback [5,6], imitation [7], and reward shaping [8]. The use of heuristics to this end has been pursued in [9], in order to speed up the action selection procedure during the learning process. Heuristics were also obtained from previously learned policies within a Case-Based Reasoning approach [10]. In fact, the (re)use of heuristics from a base of cases naturally leads to the Transfer Learning (TL) framework in machine learning [2], whose goal is to develop methods that allow the transfer of knowledge obtained in one domain to another [11].

Transfer learning is an important tool to speed up RL algorithms since, in RL, small changes on a problem configuration usually require complete new training. Within a Transfer Learning context, this complete re-training can be simplified, as the knowledge acquired in a previous situation can be re-used as heuristics, accelerating the learning procedure in the new situation.

A very informative definition of transfer learning is given in [11]: given a source domain ($\mathcal{D}_s$) with its related task ($\mathcal{T}_s$), and a target domain ($\mathcal{D}_t$) with its related task ($\mathcal{T}_t$), transfer learning aims to improve the performance of learning the task's predictive function at the *target* domain, using the knowledge obtained on learning the predictive function in $\mathcal{D}_s$ and $\mathcal{T}_s$, for $\mathcal{D}_s \neq \mathcal{D}_t$ or $\mathcal{T}_s \neq \mathcal{T}_t$. Also, according to Pan and Yang [11], transfer learning algorithms can be characterised by (1) *which* information is transfered; (2) *how* it is transfered; and, (3) *when* it is transfered. The first characteristic refers to what part of knowledge is chosen to be transfered, in RL it could be the rewards or the policy, for instance. The second feature is related to the algorithms used to transfer the knowledge from one domain to the other, which should take into account task mappings; and the third feature specifies in which situations the knowledge should be transfered.

This paper investigates the strengths of a transfer learning meta-algorithm, called L3, that provides a framework that allows the implementation of different algorithms using heuristics to accelerate a Reinforcement Learning procedure in one domain (target), that are obtained from another, simpler, domain (source). The L3 meta-algorithm works in 3 stages: first, it uses an RL algorithm to learn how to perform one task, storing the solution for this problem as a case in a case base; in the second stage, it maps actions of the source domain to actions of the target domain; and, in the last stage, it uses the stored cases as heuristics to speed up the Reinforcement Learning process in the target domain. By using a case base as part of the transfer learning procedure, L3 falls within the class of case-based transfer learning algorithms, discussed in [12]. A preliminary investigation on this meta-algorithm was presented in [13], where an L3 instance based on the Q-Learning algorithm was described. The present paper extends our previous work in three ways: first, this work introduces a new L3 algorithm based on the SARSA(λ) algorithm; second, the proposed algorithms are evaluated on more domains than those described in [13]; and, third, we show that the L3 framework is robust under the transference of *negative* information.

The theoretical background, that supports this work, is presented in Section 3, that also presents the evolution of methods that use heuristics to accelerate RL. The L3 meta-algorithm, that is built upon this theoretical background, is described in Section 4, and Section 5 presents the empirical results of applying L3 in two distinct domains: the 3D mountain car and stability learning for a humanoid robot in the Robocup 3D Soccer Simulator. The results show that L3 outperforms several algorithms, including Q-Learning, used as baseline algorithm, and two state-of-the-art transfer learning algorithms.

2. Related work

The field of transfer learning can be seen as the consolidation of a set of techniques proposed over the past years [11], such as life-long learning [14], knowledge (or inductive) transfer [15], metalearning [16], among others. There are also closely related techniques, such as multi-task learning [17], imitation learning [18] and human advice [19]. The goal of multi-task learning is to learn multiple tasks simultaneously, using common features in the pool of (distinct) tasks to accelerate the learning process of each of the tasks individually; in contrast, transfer learning focus on accelerating the learning rate of one target task only, given the knowledge obtained from learning a previous task. Imitation learning aims at speeding up the learning process of a task by using the knowledge obtained from the observation of another agent solving the same task, whereas transfer learning concentrates on the transference of the knowledge obtained by a single agent on solving distinct (but related) tasks [2]. Similarly to imitation learning, human advice integrates knowledge provided by a human agent in the machine learning loop, which is an issue outside the interest of current methods in transfer learning.

In a broader sense, transfer learning may be placed within the *abstraction in artificial intelligence* umbrella, as abstractions in AI usually relate a ground problem with a more abstract one (not necessarily a more abstract version of the same problem), according to [20]. In fact, Saitta and Zucker [20, p. 54] point out that a comprehensive theory of abstraction should provide "*the framework to support the transfer of techniques between different domains*".

The application of Transfer Learning within Reinforcement Learning tasks was first proposed in [4], where an algorithm was defined that exploits strong features obtained from RL on one task in order to compose functions in a case base that is used on the solution of a new task. More recent work on transferring cases for RL includes [21], which propose a technique for abstracting reusable cases from RL, enabling the transfer of acquired knowledge to other instances of the same problem. A method that abstracts the *intention* of an actor on solving a task, turning it into a case base for RL is proposed in [22]. Focusing on policy reuse, Fernández and Veloso [23] propose a method that uses previously learned policies as a probabilistic bias that guides the exploration/exploitation process. The principles of knowledge transfer were applied to general game learning in [24], whereby the knowledge obtained in learning one particular game is generalised to be used in other games. The problem of transferring policies across continuous domains was tackled in [25] by means of a model minimisation strategy for mapping state-action pairs. In contrast to previous work, which were mostly based on model-free methods, Taylor et al. [26] propose a transfer-learning method for a model-based reinforcement learning algorithm for continuous state space. The problem of finding the appropriate potential function for accelerating the task performance of the target domain is defined as a supervised regression problem in [27]. In this regression problem the goal is to select the features that are most relevant to the potential function by means of the features' influence on the prediction of the cross-task value. A more complete survey of transfer learning in reinforcement learning, up till 2009, is presented in [2].

It is evident in the literature of transfer learning in RL that most of the early approaches typically use hand-coded inter-task mapping for transferring knowledge from one domain to another [28]. More recently, there has been an increasing number of work reporting successful experiments on learning inter-task mapping. The work proposed by Taylor [29] was perhaps the first approach to automate the process of inter-task mappings in TL, where actions with similar effects (in the target and source domains) were associated to allow the transference of knowledge across the domains. In fact, Taylor [29] explores the use of inter-task mappings with various possibilities of features to transfer, such as *Value-function*, *Q-value*, *Policy* and *rules* representing the source-task policy learned. This work also proposes the "Modeling Approximated State Transitions by Exploiting Regression" (MASTER) algorithm, that automatically learns a mapping between source and target tasks using an agent's experience.

In [30] the correspondence between state spaces of the tasks is accomplished assuming an underlying *subspace* (called the *common-task subspace*) that relates the source and target tasks. The inter-task mapping is then autonomously determined by a function approximation technique. The common task subspace, however, is determined manually. Relaxing the need for a hand-coded subspace, Ammar et al. [28] propose a supervised method for learning the inter-task mapping by using sparse coding with a similarity measure. The use of multiple inter-task mappings in transfer learning is investigated in [31] for both model-free and model-based RL. In this work, in order to avoid negative mappings, the authors propose a method for selecting the most relevant mappings. A hybrid approach implementing model-free and model-based learning for transferring models of potential-based, reward shaping functions is proposed in [32], whereby the transition and reward functions of the source task are obtained by cascade neural networks. A value-function approximation (the Cerebral Model Articulation Controller, CMAC [33]) is used to find the value of a state, given the neighbouring state values. In the target task this algorithm generates simulations from the source task that are used in an RL step; finally, the target and source models are queried for a new state, the algorithm then chooses which model (either target or source) to use, selecting that with the least prediction error for that state.

The meta-algorithm proposed in this paper falls into the category of methods whose inter-task mapping is automatically learned. For this reason, the algorithms proposed in [32] and in [29] (mentioned above) are used in the present paper for comparison purposes, as described in Section 5.

In contrast to other work in this area, as we shall see further in this paper, in L3 the knowledge is transferred across domains in terms of heuristics that are stored as a case base to be used to accelerate reinforcement learning in the target domains. This allows the learning process to recover from negative (or imprecise) transfers.

The next section presents the theoretical background of this work, tracing the evolution of the use of heuristics to speed up RL.

3. Background

The task of an RL agent is to learn an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maps the current state s into a desirable action a to be performed in s [34]. In RL, the policy π should be learned through trial-and-error interactions of the agent with its environment. This problem is usually formulated as a discrete time, finite state, finite action Markov Decision Process (MDP), where the learner's environment is modelled as a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, in which: $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions that the agent can perform; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is a state transition function, where $\Pi(\mathcal{S})$ is a probability distribution over $\mathcal{S}$; and, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: is a reward function [35].

Within the set of RL algorithms, Q-learning [36] obtains an optimal policy π^* when the model ($\mathcal{T}$ and $\mathcal{R}$) is not known in advance. This is done by using the following update rule:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right], \tag{1}$$

where s is the current state; a is the action performed in s; r is the reward received; s' is the new state obtained by executing action a in state s; α is the learning rate ($\alpha = 1/(1 + visits(s,a))$), and γ is a discount factor ($0 \le \gamma < 1$). The term $visits(s,a)$ is the total number of times this state-action pair has been visited up to, and including, the current iteration.

The SARSA algorithm [37] is a modification of Q-learning that updates the policy based on actions taken during the interaction with the environment – this kind of learning is known as on-policy. The SARSA learning rule does not include the maximisation that exists in the Q-learning rule, and can be represented by Equation (2) below:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r + \gamma \hat{Q}(s',a') - \hat{Q}(s,a) \right], \tag{2}$$

where all the variables are defined in the same way as in Equation (1).

Algorithm 1 HAQL algorithm [39].

```
Require: The learning rate $\alpha$, the exploration/exploitation rate $\epsilon$, the reward function $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the heuristic function $\mathcal{H}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a small real
    value $\eta$ and $\xi$ used to weight the influence of the heuristic.
 1: Initialise $\hat{Q}_t(s_t, a_t)$ and $H_t(s_t, a_t)$ arbitrarily.
 2: repeat
      {for each episode}
 3:   Initialise $s_t$.
 4:   repeat
        {for each step}
 5:     Compute $H_t(s_t, a_t)$.
 6:     Select an action $a_t$ using the modified $\epsilon$-Greedy rule:
          $a_t = \begin{cases} \arg\max_{a_t} \left[ \hat{Q}(s_t, a_t) + \xi H_t(s_t, a_t) \right] & \text{if } q \le \epsilon, \\ a_{random} & \text{otherwise,} \end{cases}$
 7:     Execute the action $a_t$.
 8:     Observe $r(s_t, a_t)$, $s_{t+1}$.
 9:     $\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)]$
10:     $s_t \leftarrow s_{t+1}$.
11:   until $s_t$ is terminal.
12: until some stopping criterion is reached
```

The SARSA algorithm outperforms Q-learning when the use of exploration occasionally results in a large negative reward, learning to avoid "dangerous areas" on the learning space.

The Heuristically Accelerated Reinforcement Learning (HARL) is a class of algorithms [9] that solves the RL problem by making explicit use of a heuristic function $\mathcal{H}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to influence the choice of actions during the learning process. The heuristic function is used only in the action-choice rule; it defines which action a_t must be executed when the agent is in a state s_t. The action-choice rule used in HARL is a modification of the standard ϵ-*Greedy* rule used in Reinforcement Learning, but with the heuristic function included:

$$\pi(s_t) = \begin{cases} \arg\max_{a_t} \left[\hat{Q}(s_t, a_t) + \xi H_t(s_t, a_t) \right] & \text{if } q \le \epsilon, \\ a_{random} & \text{otherwise,} \end{cases} \tag{3}$$

where: $\mathcal{H}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the heuristic function, which influences the action choice. The subscript t indicates that the heuristic function can be non-stationary; ξ is a real variable used to weigh the influence of the heuristic function; q is a random value with uniform probability in $[0, 1]$ and ϵ $(0 \le \epsilon \le 1)$ is the parameter that defines the exploration/exploitation trade-off: the greater the value of p, the smaller is the probability of a random choice; a_{random} is a random action selected from the set of possible actions in the state s_t.

As a general rule, the value of the heuristic $H_t(s_t, a_t)$ used in the HARL must be higher than the variation among $\hat{Q}(s_t, a_t)$ for a similar $s_t \in S$ (so that it can influence the choice of actions) and it must be as low as possible in order to minimise the error. There are several possibilities to compute $H_t(s_t, a_t)$, from using a large value that is lower than $r_n/(1-\gamma)$, where r_n is the negative reward the agent receives in each time step (see [38] for a discussion on this value), to using a small value that depends on the instant values of the value function approximation, that can be defined as:

$$H_t(s_t, a_t) = \begin{cases} \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + \eta & \text{if } a_t = \pi^H(s_t), \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where η is a small real value and $\pi^H(s_t)$ is the action suggested by the heuristic H.

Bianchi et al. [38] showed that as the heuristic is used only in the choice of the action to be taken, this algorithm differs from the original RL algorithm only in the way the exploration is carried out. The only convergence condition of the RL algorithm that could be affected by the exploration made in the HARL is the necessity of infinite visitation to each pair state-action. One option to validate this condition is to recede the influence of the heuristics with time, by multiplying ξ by a decay factor. Other options is to use other visitation strategies, such as intercalating steps where the algorithm makes alternate use of the heuristics and exploration steps, or using the heuristics only during a period of time, smaller than the total learning time for RL algorithm. Thus the formal results obtained for RL algorithms remain valid for HARL.

HAQL was the first HARL algorithm implemented [39]. It extends the Q-Learning algorithm by using the heuristic function to influence the action choice. HAQL has been used in a variety of domains such as autonomous mobile robot navigation [9], RoboCup 2D Simulation [40], Multi-Robot Task Allocation (MRTA) applied in the RoboCup Small Size League [41]; it was also extended to deal with multiagent problems [42]. The HAQL algorithm is shown in Algorithm 1.

The use of heuristics in the SARSA algorithm was recently proposed by Bianchi et al. [38]. In the same paper, the authors also expand the number of HARL algorithms by proposing the Heuristically Accelerated Q(λ), HA-SARSA(λ) and HA-TD(λ), the first algorithms that used both heuristics and eligibility traces.

The HAQL algorithm was extended in [43] to allow the retrieval and reuse of heuristics from a case base. In this algorithm, called CB-HAQL (see Algorithm 2), steps were added before the action selection is made in order to compute the similarity of the cases with the current state and the cost of adaptation.

Algorithm 2 CB-HAQL algorithm [43].

```
Require: The learning rate $\alpha$, the exploration/exploitation rate $\epsilon$, the reward function $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the heuristic function $\mathcal{H}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a small real
    value $\eta$ and $\xi$ used to weight the influence of the heuristic. Also requires a case base $C$ that is different for each problem.
 1: Initialise $\hat{Q}_t(s,a)$ and $H_t(s,a)$ arbitrarily.
 2: repeat
      {for each episode}
 3:   Initialise $s_t$.
 4:   repeat
        {for each step}
 5:     Compute similarity between the current state and all the states in the case base.
 6:     Retrieve the case that is most similar to the current problem.
 7:     if the retrieved case is similar to the current state then
 8:       Compute $H_t(s_t, a_t)$.
 9:     end if
10:     Select an action $a_t$ using the modified $\epsilon$-Greedy rule:
          $$a_t = \begin{cases} \arg\max_{a_t} \left[ \hat{Q}(s_t, a_t) + \xi H_t(s_t, a_t) \right] & \text{if } q \le \epsilon, \\ a_{random} & \text{otherwise,} \end{cases}$$
11:     Execute the action $a_t$. Observe $r(s_t, a_t)$, $s_{t+1}$.
12:     $\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)]$
13:     $s_t \leftarrow s_{t+1}$.
14:   until $s_t$ is terminal.
15: until some stopping criterion is reached
```

Algorithm 3 L3 meta-algorithm.

```
    {STAGE 1: Case base construction}
 1: Use a Reinforcement Learning algorithm to compute the optimal policy for the source domain.
 2: Create a case base.
    {STAGE 2: Action-mapping across domains}
 3: Map actions from source domain to target domain using a Neural Network (Algorithm 4).
    {STAGE 3: Reusing the case base in a CB-HARL algorithm}
 4: Use the case base in a CB-HARL algorithm to solve the problem in the target domain (Algorithm 2).
```

A case is retrieved if the similarity between the new problem and a case in the case base is above a certain threshold. To compute the similarity, several functions can be used. For example, the distance between the attributes of the new problem and the problem in the case base can be computed using a distance metric such as the Manhattan distance, the Euclidean distance or the Gaussian distance. The problem of finding a good similarity function for a domain is well known in the literature of Case Based Reasoning, with several works dedicated to it [44,45].

The case definition used in the HAQL algorithm (and inherited by the work presented in this paper) was that proposed by Ros et al. [46], which is composed of three parts: (1) the problem description (P), which corresponds to the situation in which the case can be used; (2) the solution description, which is composed by the sequence of actions that each agent must perform to solve the problem; and, (3) the case scope that defines the applicability boundaries of the cases.

After a case is retrieved, the heuristic (with the sequence of actions suggested by the case selected) is computed using Equation (4). This heuristic is used for an amount of time proportional to the number of actions of the retrieved case. After this time interval, a new case can be retrieved.

Although only the CB-HAQL has been proposed before, we can infer that there is a class of algorithms of this kind, that extends all HARL algorithms by using cases as heuristics. We will call this class of algorithms CB-HARL, and any algorithm of this new class will differ from the one presented in Algorithm 2 only by the basic RL algorithm used. For example, the CB-HASARSA differs from the CB-HAQL only in the line 12, where the update rule used is Equation (2) instead of Equation (1).

4. Transferring a case base of heuristics: the L3 meta-algorithm

In this work we investigate one extension of CB-HARL algorithms towards transferring cases between learning agents across distinct domains. This extension has been defined within the L3 meta-algorithm (Algorithm 3), which works in three stages: first, the algorithm learns how to perform a task in the source domain, storing the optimal policy for this problem as a case base; second, it maps actions from the source domain to actions in the target domain; and third, it uses the case base learned in the first stage as heuristics in a CB-HARL algorithm. The L3 processing stages are detailed as follows.

Stage 1: In the case base construction phase, an RL algorithm (such as Q-learning or SARSA) is used to compute the optimal policy for the source domain. A case base is then built from the learned policy, with a pre-defined number of cases. Similar to the model proposed in [46], each case is described by a 3-tuple: $case = (P, A, R)$, where: P is the description of the problem, containing all relevant information of the agent state; A is an action that solves the problem; and, R is the expected return for performing the action, which indicates the quality of the action stored in this case. There are several ways by which the case-base can be built from the learned policy, such as:

Algorithm 4 Action-mapping across domains.

```
1: for a large number of iterations do
2:     Randomly select an action in the Source domain;
3:     Randomly select an action in the Target domain;
4:     Execute the selected actions in both domains;
5:     Compute the consequences of the executed action in all the dimensions of the Source domain;
6:     Compute the consequences of the executed action in all the dimensions of the Target domain;
7:     Update the connection between neurons of the selected actions using Equation (5);
8: end for
```

- select N cases by random sampling the state set and finding the best action for that state;
- Select N cases by random sampling the action-state set and excluding the cases that contains the worst actions for the state chosen;
- Select N cases that have the best Q value.

These cases will be explored in the experiments below.

Stage 2: In the action-mapping stage, a Neural Network maps the actions between the source task and the target task. In this network, the input nodes correspond to the set of possible actions in the target domain, and the output nodes correspond to the set of actions of the source domain. In order to learn the network weights, a set of random actions is executed in both domains. If the observed results of the two actions are similar (for example, both actions lead to an increase in the x speed of a robot), the weight that links this pair of actions is increased. If the results of the two actions are different, the weight of the connection is decreased. In this way, this neural network learns the relationship between the consequences of actions in both domains.

This scheme can be formalised as a single layer, forward-feed, unsupervised neural network using the Hebbian learning rule, where the input and output vectors are in bipolar form (-1 or 1). The Hebbian rule (or Hebbian law) was proposed by Hebb [47], and it can be paraphrased as "*cells that fire together wire together; cells that fire out of sync, loose their link*".[5] The main idea of this learning rule is that the weights that connect two neurons should be increased when their outputs are similar, and decreased when they are dissimilar. This rule allows the construction of unsupervised neural networks, since it facilitates the input of training pairs that are not known *a priori*. It differs from some of the better known learning rules, such as the Delta Learning rule of the Backpropagation Algorithm [49], since the target output is not necessary for the learning process to be successful.

The Hebbian learning rule can be defined by the following equation:

$$\Delta w_{i,j} = \alpha x_i y_j, \tag{5}$$

where α is the learning rate, x_i is the input and y_j is the output neuron.

In this work, as the main goal of the neural network is to learn the relationship between the consequences of the actions in two different domains, the values of x_i and y_j used during the learning phase are the action consequences, that is, the changes in the domains due to the action executions. Algorithm 4 details the action-map learning process.

After the learning phase, the trained neural network is used to map the actions from one domain to the other. As the proposed neural network is single layer, forward feed, the output of one neuron depends on the weighted sum of the input neurons:

$$sum_j = \beta_0 + \sum_i w_{i,j} x_i, \tag{6}$$

where x is the input vector defining the target domain action, sum_j is the weighted sum and β_0 is the neuron bias. The output of the neural network is given by applying a binary activation function, Heaviside step function, to the output of all neurons:

$$y_j = \begin{cases} +1, & \text{if } sum_j \geq 0 \\ -1, & \text{if } sum_j < 0. \end{cases} \tag{7}$$

The result of this computation is a table that describes the relationship between the actions of both domains.

The benefits of this mapping approach is that it is simple, fast and effective. However, this requires that the state mapping is given so that the neural network could be applied. Future work shall consider developing a mapping procedure capable of both, mapping states and actions autonomously.

Stage 3: In the final stage, the previously stored case base is used in a CB-HARL algorithm to speed up task learning in the target domain.

Case retrieval (in this context) is driven by a similarity measure between the new problem and the data in the case base. Inspired in the case retrieval method proposed in [46], in this work we use a similarity function to compute the similarity between the new problem and the stored case base:

[5] This mnemonic sentence is attributed to Carla Shatz at Stanford University [48].

$$Sim(P,C) = \sum_{i=0}^{n} dist(A_i{}^c, A_i{}^p), \tag{8}$$

where $A_i{}^c$ is the value of the attribute i in the description of the case, $A_i{}^p$ is its value in the new problem, and $dist(a,b)$ is the distance between objects a and b. As described in Section 3, the distance between the attributes of the new problem and the problem in the case base can be computed using a distance metric such as the Manhattan distance, the Euclidean distance or the Gaussian distance. To retrieve a case, the similarity between all the cases in the case base and the new problem are computed, and the case that is most similar to the new problem is retrieved.

After a case is retrieved, a heuristic is computed using Equation (4) and the action suggested by the case is selected and executed. If the case base does not contain a case that can be used in the current situation (i.e., the similarity between all the cases in the case base and the current situation is below a pre-defined threshold), the CB-HARL algorithm behaves as the original RL algorithm implemented in it.

Using the L3 framework, two algorithms were implemented in this work:

- L3-Q, which is based on the CB-HAQL, and was previously proposed by Celiberto et al. [13], and
- L3-SARSA(λ), which extends the SARSA(λ) algorithm to include the use of cases as heuristics in the action selection.

5. Results and evaluation

In this section we present experiments where two L3 algorithms, L3-Q and L3-SARSA(λ), are applied in two domains: one which is a traditional benchmark in Reinforcement Learning, the Mountain Car Problem; and a more recent control problem, the Stabilisation of a Humanoid Robot. The first experiment shows how a control strategy can be mapped and transferred between similar domains and the last experiment shows a more complex transfer procedure. Our claim here is that L3 outperforms its non-transfer learning version, and also state-of-the-art transfer learning algorithms.

In [2] several performance metrics for transfer learning were proposed. Although each one of these metrics have known drawbacks, and others can be proposed, these five metrics are becoming a standard in the transfer learning community. The metrics are:

1. *Jumpstart*: the initial improvement of the performance of an agent in the target task, given by the transfer;
2. *Asymptotic performance*: the final learned performance of the agent in the target task;
3. *Total reward*: the total reward earned by the agent;
4. *Transfer ratio*: the ratio between the total reward received by an agent that used the transfer and the one received by an agent that does not use transfer learning;
5. *Time to threshold*: the time taken by an agent to achieve a pre-defined level of performance.

These metrics were used in the experiments below to provide more information for comparing the algorithms in a multi-dimensional evaluation procedure.

Student's t-Test [50] was also used in this work as a statistical test to verify the hypothesis that L3 speeds up the learning process. According to Nehmzow [51], if two different control programs produce two distinct means of a particular result, the t-Test can be used to decide whether there is a significant difference between these two means. The greater the value of T, the more significantly different are the results.

In order to show that the L3 meta-algorithm improves the learning rate of the system, we compare the obtained results with those obtained with other RL, HARL and TL algorithms.

5.1. Experiment 1: mountain car problem

In this experiment we tested how cases acquired in the 2D mountain car problem [52] can be transferred and used to speed up the learning in the 3D mountain car problem [53].

In the mountain car problem, a car that is located at the bottom of a valley is moved backward and forward until it reaches the top of a hill. The goal of the learning agent is to generalise across continuous state variables in order to learn how to drive the car up to the goal state.

In the 2D mountain car problem two continuous variables describe the agent's state: the horizontal position (x) restricted to the range $[-1.2, 0.6]$ and the velocity ($\dot{x}$) restricted to the range $[-0.07, 0.07]$. The agent may select one of the following three actions on each step:

$$\{left(-1), neutral(0), right(+1)\},$$

which change the velocity by -0.0007, 0, and 0.0007 respectively. The car reaches the top of the hill when its horizontal position is greater than 0.5. The system dynamics is given by Equations (9) below:

$$\begin{aligned} x_{t+1} &= \min(0.5, \max(-1.2, x_t + \dot{x}_{t+1})); \\ \dot{x}_{t+1} &= \min(0.07, \max(-0.07, \dot{x}_t + 0.001a_t - 0.0025cos(3x_t))). \end{aligned} \tag{9}$$

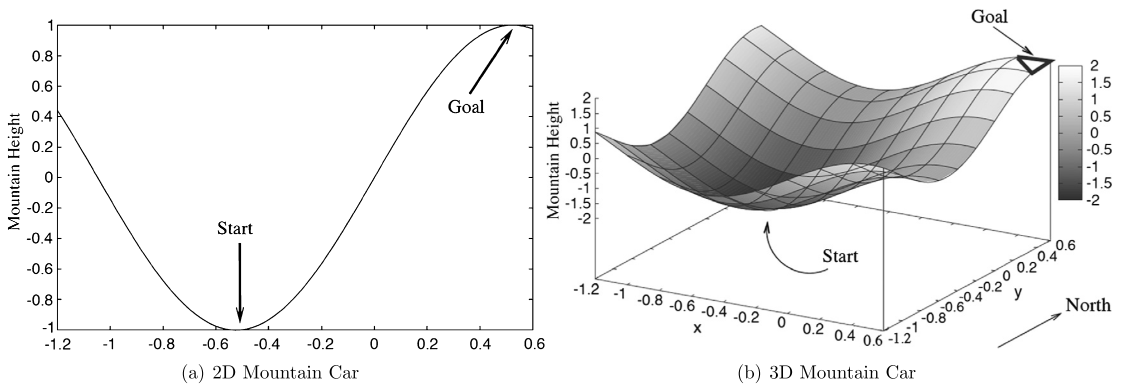


Fig. 1. Two versions of the Mountain Car Problem (image adapted from Tayor [29]).

The 3D mountain car is similar to its 2D version, whereas the former is defined over a surface rather than on a curve [29] (as shown in Fig. 1). The state is composed of four continuous state variables: x, $\dot{x}$, y, $\dot{y}$. The positions and velocities have ranges of $[-1.2, 0.6]$ and $[-0.07, 0.07]$, respectively. The agent can select from five actions at each time step: $\{neutral, west, east, south, north\}$. The actions *west* and *east* modify $\dot{x}$ by -0.0007 and $+0.0007$ respectively, while *south* and *north* modify $\dot{y}$ by -0.0007 and $+0.0007$ respectively.

The 2D Mountain Car implementation used in this work is that described in [1] and it is available for download at [54]. This simulator implements a Gradient-Descent SARSA(λ) algorithm, using a CMAC function approximator with 9 by 9 tilings.

In the present experiment we modified the 2D Mountain Car simulator to test the results of using Q-Learning, Q(λ), SARSA, and SARSA(λ) algorithms. All of these four algorithms reached the same optimal policy. However, best results were achieved by SARSA(λ) with the parameters proposed in [1], which were: $\alpha = 0,5$, the exploration/ exploitation rate $= 0.0$, $\gamma = 1.0$. The reward is -1 on every time step until the car reaches the goal position, ending one episode.

In order to acquire the case base to be used by the L3-SARSA(λ) algorithm, SARSA(λ) was used to learn the 2D mountain car problem during 200 episodes (each episode ends either after 10.000 steps or when the agent finds the goal state). As observed in [1], this simulator could learn a near optimal policy within 100 episodes. In this work we tested with a different number of episodes, verifying that within 200 episodes learning stabilises ($\hat{Q}(s', a') - \hat{Q}(s, a) \sim 0$).

Case acquisition begins after the learning stabilises. Each case contains the state, position and velocity, (P), the action taken (A) and the expected return (R). For instance, an acquired case has the following form: $\langle P = (-1.036143, 0.013455)$; $A = right$; $R = -79.531860\rangle$.

There are several ways in which the case base acquisition can be accomplished. We tested the following four possibilities:

- Select N cases by random sampling the state set and finding the best action for that state. To do this, we define the position and speed using a random number generator, then we find the action that maximises this state ($argmax(Q)$), and its Q value, saving this data as a case. In this way, the case base is built using the best actions for the 2D problem, for a uniformly distributed set of states.
- Select N cases by random sampling the action-state set. During this sampling process, if a case contains the worst action for that state (i.e., the one with the lowest Q value), this case is discarded. In this way, the case base is built using actions that are not the worst for the 2D problem.
- Select N cases that have the best Q value. To do this, we sample $N * 1000$ times the state set, finding the best action and its Q value for a sampled state. Then, only the N cases with the highest values of Q are selected to be included in the case base. By doing this, we built a case-base with the N best state-action pairs.
- For comparison purposes, we also built a set with the N worst cases, obtained analogously to the first possibility cited above, but using $argmin(Q)$ instead of $argmax(Q)$.

The random number generator used to acquire the case base is the Mersenne Twist Random Generator [55]. This is the most widely used pseudorandom number generator up to date.

In order to use a case in the target domain, an action mapping between the two domains is needed. As described in Section 4, this mapping is made using an unsupervised neural network using the Hebbian learning rule, where the input and output vectors are in bipolar form (-1 or 1) and the activation function of the output nodes is the Heaviside step function. The network has a five-neuron input layer (one neuron for each of the 5 actions in the target domain) and three neurons in the output layer (one for each of the actions in the source domain).

In this experiment, the consequences of each action are measured as the variation of the car's speed in each dimension. To do this we compute, for all dimensions in both domains, the variations as given by:

$$variation_d = velocity_d - oldVelocity_d; \tag{10}$$

where $oldVelocity_d$ is updated at the end of each iteration, and d is the dimension.

Table 1
Action map for the Mountain Car Problem.

3D	2D
neutral	*neutral*
north	*right*
east	*right*
south	*left*
west	*left*

As the actions in the 3D Mountain Car domain are only applied in one dimension (e.g. the action *south* changes the speed only in the y direction), in order to use the action's effects to train the neural network, the dimension in which the action is applied has to be defined. This is accomplished by selecting the greatest value of the variations in the x and y axis computed for this domain. Note that z is a function of x and y.

The 3D Mountain Car implementation used in this work is based on the one described in [31], and that is available for download at [56]. The parameters used in the neural networks were: learning rate $= 0.9$, Bias $= 0.01$ and 100,000 iterations were used. The bias was used because there are changes in the velocity when using the neutral action due to the gravity force that acts on the car. The bias eliminate these effects on the mapping. One problem with this mapping is that it cannot map the *neutral* action, as it does not have any direct consequence on the target and source domains. But it can be inferred, as it is the action that has the lowest weights in the neural network connections, and that is not mapped to any other action on the source domain.

Table 1 shows the results of the automatic action-mapping executed in the second stage of the L3 algorithm. Note that the actions that accelerate the car towards the goal were mapped together, as well as the actions that accelerate the car away from the goal. The *neutral* action is inferred as the one that did not have any effect on the target domain.

The last task is now to transfer the cases to the target domain. As mentioned above, the 3D Mountain Car experiments were conducted using the simulator provided by Sutton and Barto [1], with the implementation of the Mountain Car 3D provided by Partalas [56]. The simulator was modified to introduce the L3-SARSA(λ) algorithm, with the addition of the case retrieval procedure and the use of the heuristic function.

To implement the case retrieval procedure, for each action selected, Equation (8) was used to compute the similarity between the current state of the car and each case in the case base. As the source task has only two attributes (horizontal position x and velocity $\dot{x}$) in the case base, and the problem has four attributes (x, $\dot{x}$, y, $\dot{y}$), we used Equation (8) to find the most similar case between each degree of freedom of the 3D problem and the 2D problem, and computed the similarity of a case as the minimum value between these two results, as stated in Equation (11):

$$Sim(P,C) = Min[dist({x_i}^c, x^p) + dist(\dot{x}_i^c, \dot{x}^p), dist({x_i}^c, y^p) + dist(\dot{x}_i^c, \dot{y}^p)], \tag{11}$$

where $dist(a,b) = |a-b|$ is the Manhattan Distance between two points.

The heuristic used in the L3-SARSA(λ) was computed using Equation (4), where s_t is the current state and a_t is equivalent to a^c, which is the action suggested by the most similar case in the case base. To compare both actions, a_t must be mapped to the source domain using the learned mapping function $map : A_t \to A_s$, which maps actions from the target domain into actions in the source domain. Equation (4) can be rewritten as:

$$H_t(s_t,a_t) = \begin{cases} \max_a \hat{Q}(s_t,a) - \hat{Q}(s_t,a_t) + \eta & \text{if } map(a_t) = a^c, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Thirty training sessions (each of which contained 1000 episodes) were executed for four algorithms in the 3D domain: SARSA(λ), HA-SARSA(λ), L3-SARSA(λ) and the TiMRLA Value-Addition algorithm (described in Fachantidis et al. [31]). For comparison purposes we also include the results obtained by Taylor's MASTER Algorithm, published in [29, Chapter 7].

The last two are state-of-the art transfer learning algorithms that were selected for comparison purposes (as mentioned in Section 2).

The heuristic used in HA-SARSA(λ) was defined by a simple rule: if the velocity is negative, use a negative thrust, i.e., decrease the velocity by 0.0007. This heuristic can be expressed by:

$$H_t(s_t,a_t) = \begin{cases} +100 & \text{if } \dot{x}_t < 0 \,\&\, a_t = west, \\ +100 & \text{if } \dot{y}_t < 0 \,\&\, a_t = south, \\ 0 & \text{otherwise,} \end{cases} \tag{13}$$

where $\dot{x}_t$ and $\dot{y}_t$ are elements of the state s_t.

The TiMRLA Value-Addition algorithm described in [31] was executed using the software provided by the authors (available in [57]). The case base used in L3-SARSA(λ) was the one that contains N cases selected by random sampling the state set and finding the best action for that state, because this is the case base that produced the best results for the transfer.[6]

[6] A comparison between the case bases are shown at the end of this section (cf. Fig. 5).

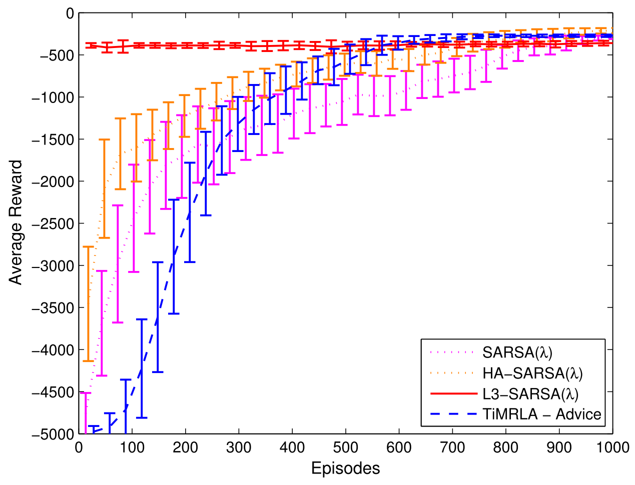


Fig. 2. The learning curves for SARSA(λ), HA-SARSA(λ), the TiMRLA Value-Addition algorithm and L3-SARSA(λ) in the 3D Mountain Car Problem. The curves are smoothed with a window of 25 episodes with an error bar at every 25 episodes.

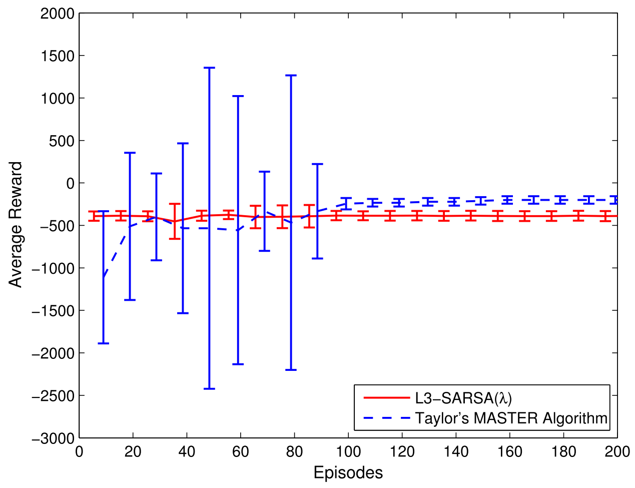


Fig. 3. The learning curves for L3-SARSA(λ) and Taylor's MASTER algorithm in the 3D Mountain Car Problem. The curve for L3-SARSA(λ) is smoothed with a window of 10 episodes with an error bar at every 10 episodes. Taylor's MASTER Algorithm results were extracted from [29, Fig. 7.6, p. 226].

The parameters used in the experiments were the same over all trials: the learning rate $\alpha = 0.2$, the exploration/exploitation rate $\epsilon = 0.0$, $\lambda = 0.95$, $\gamma = 1.0$. The parameter used to create the heuristics in L3 is $\eta = 1$ and the parameter that weights the influence of the heuristic $\xi = 1$, with a decay of 10^{-4} at the end of each episode. Values in the Q table were randomly initiated. Fig. 2 shows the learning curves obtained, where it can be noted that L3-SARSA(λ) outperforms all other algorithms until episode 600; after that the performances of the algorithms converge. The curves for L3-SARSA(λ) and Taylor's MASTER algorithm are shown in Fig. 3.

For the experiments reported in this section, the value of the module of T (of the t-Student test) was computed for each episode using the data presented in Fig. 2. The dashed line indicates the 99% confidence limit, i.e. results above the line are different and the probability for this statement to be erroneous is 1%. The results in Fig. 4 show that L3-SARSA(λ) performs clearly better than SARSA(λ) until the 750th episode, HA-SARSA(λ) until the 450th episode with a level of confidence greater than 99%. It also outperforms the TiMRLA Value-Addition algorithm until the 450th episode. After that the performances of all algorithms are equivalent, as expected.

The results of the comparison between L3-SARSA(λ) and Taylor's MASTER algorithm (Fig. 3) shows that L3 outperforms MASTER in the initial episodes. After the 20th episode, the errorbars on MASTER's results are very large, making the value of T drop below the 99% limit in which there is confidence that the algorithms are different. The asymptotic values of the MASTER algorithm are higher than L3-SARSA(λ), but they are still within the limit in which it can be said that the performances are the same.

Tables 2 and 3 show the values of *Jumpstart, Asymptotic Performance, Total Reward, Transfer Ratio* and *Time To Threshold* metrics for the algorithms executed in this experiment. These tables show that L3-SARSA(λ) outperforms the other algorithms. The closest results to those obtained by L3-SARSA were obtained on Asymptotic Performance of the MASTER algorithm. In

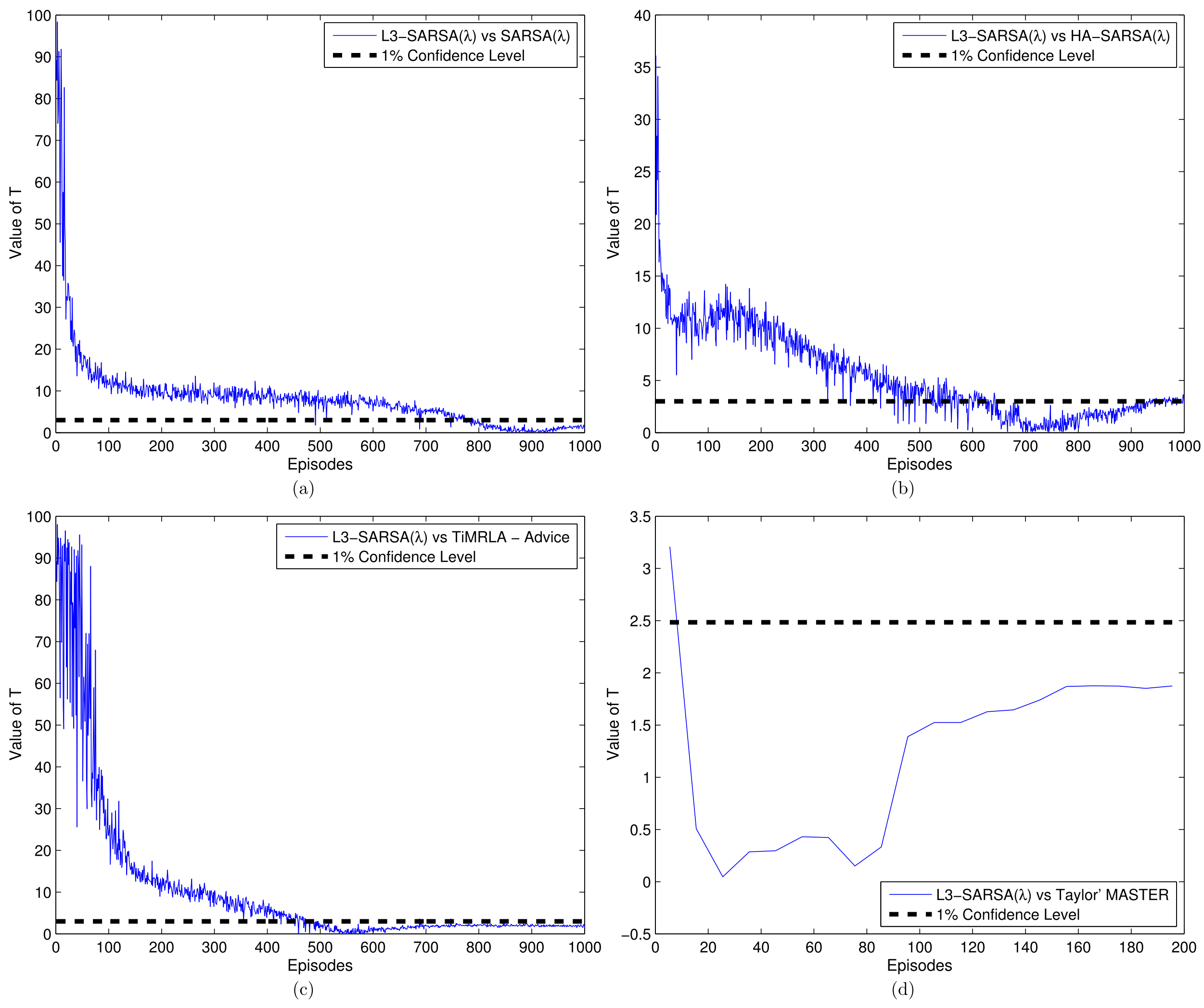


Fig. 4. Student's t-test between (a) L3-SARSA(λ) and SARSA(λ), (b) between L3-SARSA(λ) and HA-SARSA(λ), (c) L3-SARSA(λ) and the TiMRLA Value-Addition algorithm by Fachantidis et al. [31] and (d) L3-SARSA(λ) and MASTER algorithm by Taylor [29].

Table 2
Jumpstart improvement, Asymptotic Performance, Total Reward and Transfer Ratio for the SARSA(λ), HA-SARSA(λ), L3-SARSA(λ), TiMRLA Value-Addition algorithm by Fachantidis et al. [31] and MASTER algorithm by Taylor [29].

Algorithm	Jumpstart	Asymptotic performance	Total reward ($\times 10^3$)	Transfer ratio
SARSA(λ)	–	-287 ± 16	-1091	–
HA-SARSA(λ)	455 ± 397	-205 ± 24	-692	1.57
L3-SARSA(λ)	4600 ± 60	-250 ± 27	-381	2.86
TiMRLA	0 ± 0	-271 ± 14	-1054	1.05
MASTER	3889 ± 780	-200 ± 45	-231	4.72

fact, MASTER has a higher mean value of the Asymptotic Performance (and a higher Total Reward as a consequence) than those of L3-SARSA, however the t-Student test shows that results from MASTER and L3-SARSA are not statistically distinct.

In order to decide which case base should be used in the transfer, from the three possibilities acquired (as mentioned above), we compared the results of using each case base with the L3-SARSA(λ) algorithm in the target domain. The results, presented in Fig. 5, show that the best case base is the one that selects N cases by random sampling the state set, finding the best action for that state. For this experiment, the value of the module of T (of the t-Student test) was computed for each episode using the data represented in Fig. 5. The results are similar to the ones in Fig. 4 and show that L3-SARSA(λ), using the cases with the best actions from random sampling of the state set, outperforms all the other case bases until the

Table 3
Time to threshold for the SARSA(λ), HA-SARSA(λ), L3-SARSA(λ) and TiMRLA Value-Addition algorithm by Fachantidis et al. [31] and MASTER algorithm by Taylor [29].

Algorithm	Time to threshold (episodes)				
	−4000	−3000	−2000	−1000	−400
SARSA(λ)	33	77	140	508	868
HA-SARSA(λ)	6	17	49	256	709
L3-SARSA(λ)	0	0	0	0	0
TiMRLA	117	157	216	345	508
MASTER	0	0	0	10	28

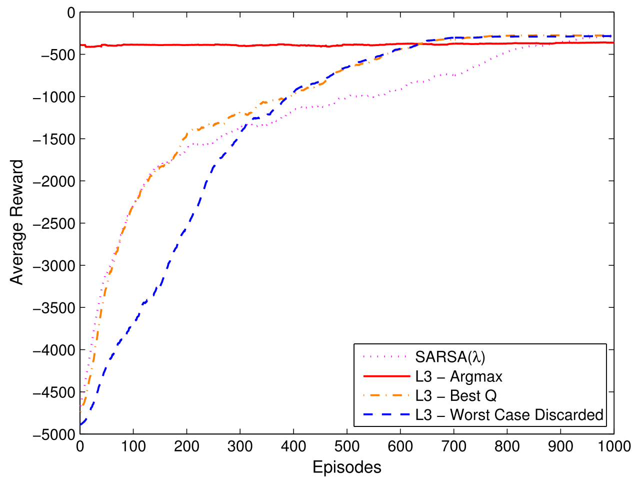


Fig. 5. The learning curves for SARSA(λ) and L3-SARSA(λ) using three different case bases in the 3D Mountain Car Problem. The curves are smoothed with a window of 25 episodes.

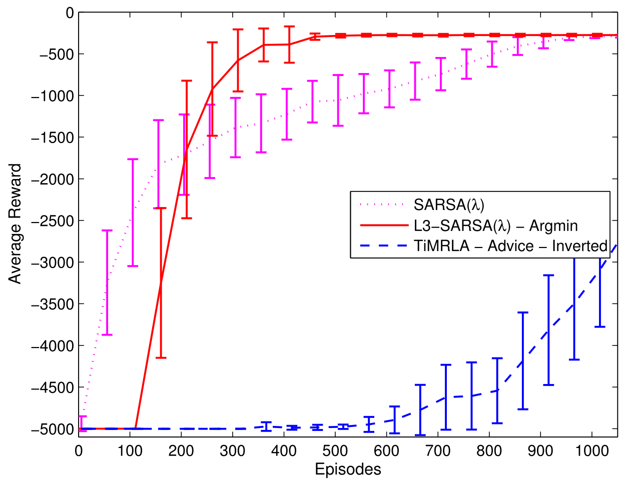


Fig. 6. The learning curves for SARSA(λ) and L3-SARSA(λ) using the case base constructed with the *N* worst cases and TiMRLA using a Negative Transfer in the 3D Mountain Car Problem. The curves are smoothed with a window of 25 episodes.

600th episode, when they reach the same performance level, with a level of confidence greater than 99%. The error bars were not added in Fig. 5 in order to enhance visibility of the curves, they have the same size of those shown in Fig. 2.

The behaviour of L3-SARSA(λ) with respect to negative transfer was also investigated in this work. A negative transfer is a situation when the transfer degrades the learning agent's performance. This can be caused by problems related to the knowledge acquisition phase in the source task or to eventual incorrect mappings between the tasks.

A negative transfer was provided to L3-SARSA(λ) from a case base constructed with the *N* worst cases (as shown in Fig. 6). In order to compare the performance of our algorithm against other TL algorithms in this context, we used the TiMRLA Value-Addition algorithm with negative information obtained by inverting the values of the transfered data. The results of the negative transfer for both L3-SARSA(λ) and TiMRLA Value-Addition algorithm are shown in Fig. 6.

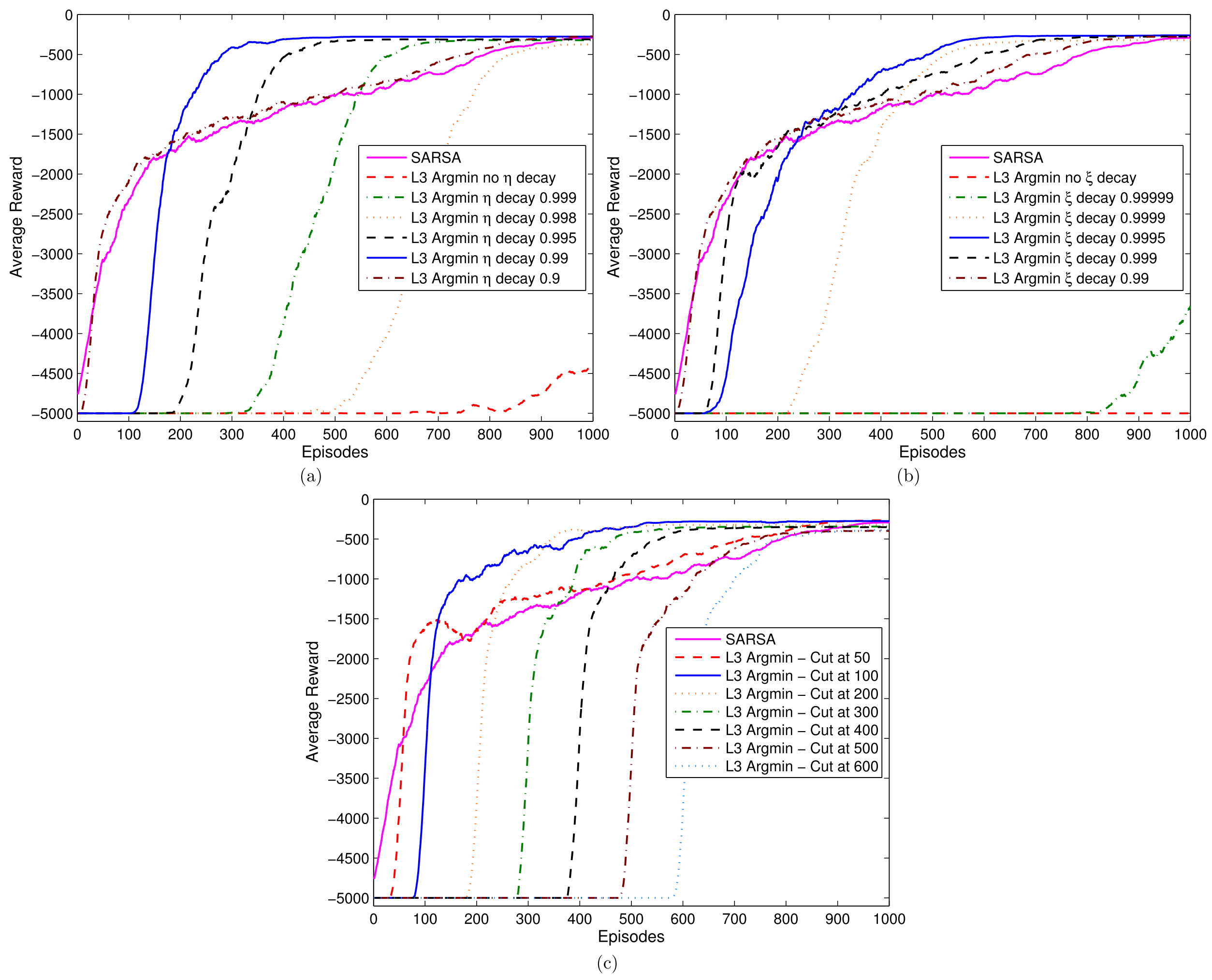


Fig. 7. The learning curves for SARSA(λ), and L3-SARSA(λ) using the case base constructed with the N worst cases, with different parameter settings, in the 3D Mountain Car Problem. (a) Uses a decay on the value of η, while (b) decays the value of ξ. (c) Makes uses of the heuristic only up to an episode number, and no heuristic after that limit. The curves are smoothed with a window of 25 episodes.

In this figure it can be seen that the negative transfer in L3-SARSA(λ) degrades the algorithm's performance until the 100th episode, after that point it learns at a fast pace, converging to the solution earlier than SARSA(λ). This behaviour is due to the fact that, as the case base contains the worst actions, at the beginning of the learning trial our algorithm only makes bad choices. But, as the case base is used as an heuristic, the SARSA(λ) algorithm learns that the actions suggested by the heuristics are misleading, and that they should not be executed. So, at this point (around the 200th episode), it begins to learn which is the best action to be performed, already knowing which are the worst actions (and not selecting any of the latter). In contrast, the TiMRLA Value-Addition algorithm with a negative transfer (shown in Fig. 6) had a very poor asymptotic performance, much lower than L3-SARSA(λ) with negative transfer and the traditional SARSA(λ) algorithm.

In Section 3, we stated that the convergence of the HARL algorithms depends on the infinite visitation to each pair state-action condition, and that this condition can be considered valid for these algorithms by reducing the influence of the heuristics over time in a number of manners: by multiplying ξ by a decay factor; by multiplying η by a decaying factor; or by using the heuristics only during a period of time, shorter than the total learning time for RL algorithm.

Fig. 7 shows that the results of the negative transfer when using L3-SARSA(λ) depends on the value of the η and ξ parameters and their decay (Fig. 7a and b), and in the number of episodes that the heuristic is used (Fig. 7c). Bianchi et al. [58] showed that using a fixed value for η and ξ, the algorithm takes longer to ignore the negative transfer. Multiplying ξ by a decay value at the end of each episode reduces the influence of the heuristics over time. If this value is too high, the algorithm takes longer to ignore the misleading heuristics. On the other hand, if this value is too low, the algorithm ignores the heuristics too soon, which makes it behave as the traditional SARSA(λ) algorithm. The same effect happens when the heuristic is used only until a certain episode (Fig. 7c). The results presented here corroborates the results presented in [58].

Table 4
The average Learning Time for SARSA(λ), HA-SARSA(λ) and L3-SARSA(λ).

Algorithm	Learning time (seconds)		
SARSA(λ)		18.2 ± 0.8	
HA-SARSA(λ)		15.7 ± 1.5	
L3-SARSA(λ)		9.29 ± 0.08	
	Case acquisition	*Mapping*	*Learning on target*
	0.11 ± 0.003	0.03 ± 0.0005	9.25 ± 0.07

Table 5
Action map for the humanoid robot stabilisation problem.

Acrobot	Robocup 3D
Positive torque	$+0.5°$ *Hip Pitch*
Negative torque	$-0.5°$ *Hip Pitch*
No torque	*No action*

This experiment was coded in C++, compiled with GNU g++, and executed on a Virtual Machine running Linux Ubuntu 14 LTS, virtualised using VM-Ware Player, and running on a MacPro running Mac OS X 10.6, 2,66 GHz Intel Xeon processor and 12 Gb of RAM memory.[7] The time needed by each part of this experiment to be executed is presented in Table 4 (where the learning time of L3-SARSA(λ) was divided into its constituting parts). It can be seen that the time to acquire the case base, and to map the source and target domains is negligible in comparison to the time needed to learn the target task. It is not possible to compare these results with the TiMRLA, because the software provided by the authors uses the deprecated RL-Glue Library [59], which makes its running time much slower than the other algorithms.

5.2. Experiment 2: humanoid robot stabilisation

In this experiment we investigate the transfer of cases acquired in the Acrobot domain [1] to accelerate stability learning for a humanoid robot in the Robocup 3D Soccer Simulator domain.

The Acrobot is a two-link pendulum operating in a vertical plane. The first joint of this actuator is passive, whereas a motor is mounted at the second joint (between the links) to provide a torque input. This system has four continuous state variables: the two joint positions, θ_1 and θ_2, and the two joint velocities $\dot{\theta_1}$ and $\dot{\theta_2}$. The goal is to swing the endpoint of the pendulum above its base by an amount equal to the equilibrium position $(\theta_1 = (\pi/2), \theta_2 = 0)$, starting from the initial state $\theta_1 = \theta_2 = 0$. There are three possible actions in this system: *positive torque*, *negative torque*, and *no torque*.

The RoboCup 3D Simulated Soccer [60] is a realistic simulator that allows the development of control techniques for humanoid robots. The current robot model used in the simulator is based on the Nao Robot by Aldebaran Robotics, which is a humanoid with 22 degrees of freedom, 57 cm high and weighing around 4.5 kg. Nao is equipped with various sensors and effectors, some of them, reproduced in the simulator, are: angle sensors in each joint, a gyroscope, an accelerometer and a force sensor which provides information about the force applied upon the sole of each of the robot's feet.

In this experiment only three of the Nao's joints were used: *Hip Pitch*, *Knee Pitch*, and *Foot Pitch* (with left and right joint having the same position). All the other joints of the robot were kept in a stationary position. At each time step, the robot could use one of the following seven actions: $+0.5°$ *Hip Pitch*, $+0.5°$ *Knee Pitch*, $+0.5°$ *Foot Pitch*, $-0.5°$ *Hip Pitch*, $-0.5°$ *Knee Pitch*, $-0.5°$ *Foot Pitch* or *no action*. The robot starts a trial at a random position close to the equilibrium (i.e., the body leaning forward or backward in angles between -20 and 20 degrees in the foot joint). Informally, looking at the humanoid's left side a movement that makes the robot to lean forward is an anti-clockwise rotation of the robot's joints, which is a positive rotation (negative is defined on an analogous way).

In the first stage of L3-Q, Q-learning is applied in the Acrobot domain over 10,000 episodes (each episode ending either after 20.000 steps or when the agent finds the goal state). The case base acquisition starts when learning stabilises (i.e., when $\hat{Q}(s', a') - \hat{Q}(s, a) \sim 0$) which happens near the 9000th episode. From that point, 500 cases are acquired by randomly sampling the action-state set. During this sampling, cases containing actions with the lowest Q value for that state (the worst one) were discarded. In the second stage of L3-Q, the actions between the two domains are connected. This procedure mapped Acrobot's θ_1 angle with the movement of Nao's ankle (foot pitch) and Acrobot's θ_2 angle with the movement of Nao's knee. Then, the forward-feed perceptron neural network described in Section 4 is used, with input nodes corresponding to the seven actions used in the Robocup 3D Simulator (the set of possible actions in the target domain), and output nodes corresponding to the three possible actions in the Acrobot domain (the set of actions of the source domain). Table 5 shows the results of this automatic action-mapping.

[7] This virtual machine, and all the software needed to run this experiment, is available at http://fei.edu.br/~rbianchi/software.

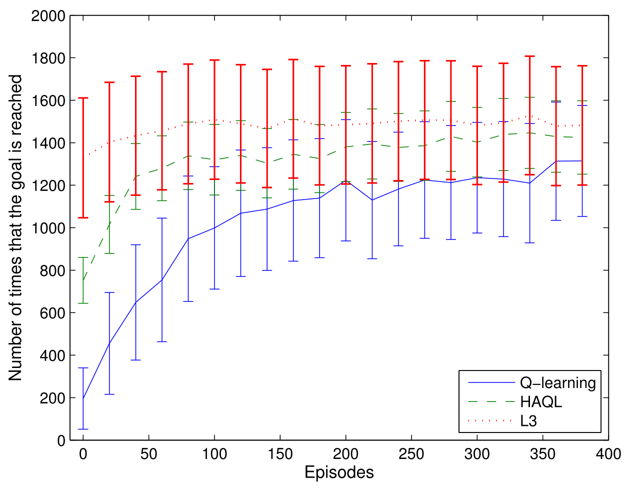


Fig. 8. The learning curves for Q-learning, HAQL and L3-Q for the Humanoid Robot Stabilisation Problem.

At the last stage of the L3-Q algorithm, the case base is used in the CB-HAQL algorithm to learn Nao's equilibrium position. The features used to compute the distance between a case and the problem are the joint angles (assumed as states in both domains). In this experiment, the distance function is defined as the Gaussian distance between attributes of the source domain and target domain:

$$dist(a,b) = exp\left(-\left[\left(\frac{a_x - b_x}{\tau^x}\right)^2 + \left(\frac{a_y - b_y}{\tau^y}\right)^2\right]\right), \tag{14}$$

where τ^x, τ^y are the radius of the scope around the object. The Gaussian distance is used because the larger the distance between two points, the lower is the similarity between them. Also, the τ^x, τ^y parameters are used as thresholds that define a maximum distance allowed for two points to have some degree of similarity, if the distance is greater than a limit, for two object's a and b, then $Sim(a,b) = 0$.

To verify that L3-Q improves the learning rate, the L3 learning curves are compared with those of Q-learning and the HAQL. This comparison was accomplished over thirty training sessions for each of the three algorithms, each of these sessions consists of 400 episodes of 120 seconds each. The same parameters were used throughout these experiments, as follows: $\alpha = 0.25$, $\gamma = 0.9$, exploration/exploitation rate $= 0.1$ and the Q table was initialised with zeroes. HAQL and L3-Q use $\eta = 1$ and $\xi = 1$, and reward of -1 on all steps which do not lead to the goal. The goal state is rewarded with $+1$. The heuristic used in the HAQL algorithm was defined using a simple rule: if the Nao is leaning forward, move the HIP angle $-0.5°$, if it is leaning backward, move the HIP angle $+0.5°$.

This is computed using the equation below:

$$H_t(s_t, a_t) = \begin{cases} \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + 1 & \text{if } HipPitch > 0 \,\&\, a_t = -0.5°, \\ \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + 1 & \text{if } HipPitch < 0 \,\&\, a_t = +0.5°, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

The results obtained are presented in Fig. 8, which shows the number of times *per* episode taken by the agent to reach the goal. It can be seen that L3-Q outperforms both Q-learning and HAQL in the initial learning phase; while the three algorithms converge to similar performance results in later episodes, as expected. The results of applying the Student's t-Test are presented in Fig. 9, where we can see that the performance obtained for L3-Q is statistically distinct from the performances of Q-learning up to the 350th episode and for HAQL up to the 50th episode, with a level of confidence greater than 1%. After that, the results are statistically indistinguishable.

One issue worth discussing in this experiment is the selection of the robot's joint whose transfer would be more effective. To transfer the learning from Acrobot to the humanoid robot three different joints could be used: the feet joint, the knee joint or the hip joint. In order to verify in which of these the transfer would produce the best results, we repeated the experiment transferring the learning each time to a different joint. The results of this comparison, presented in Fig. 10, show that the best transfer occurs at the hip joint. The t-Test applied on this problem (Fig. 11) shows that the use of L3-Q at the Hip joint is better than its application at the robot's feet up till the 100th episode, and that it was better than L3-Q applied to the robot's Knee only at the beginning of the learning trial. Future work shall address an automatic way of finding the best joint to be used in such cases.

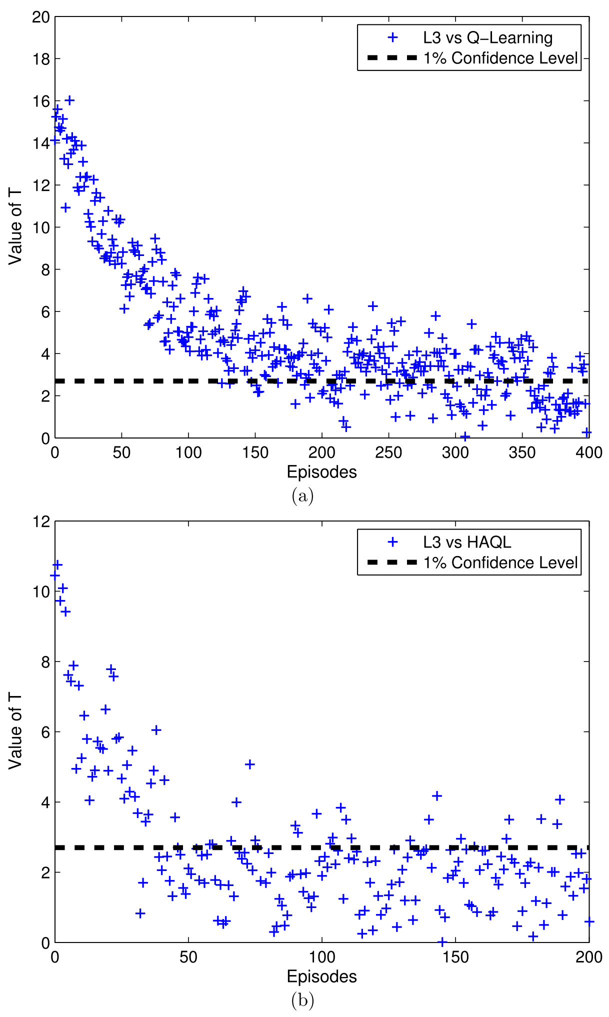


Fig. 9. Student's t-test between L3 and Q-learning (a) and between L3 and HAQL (b).

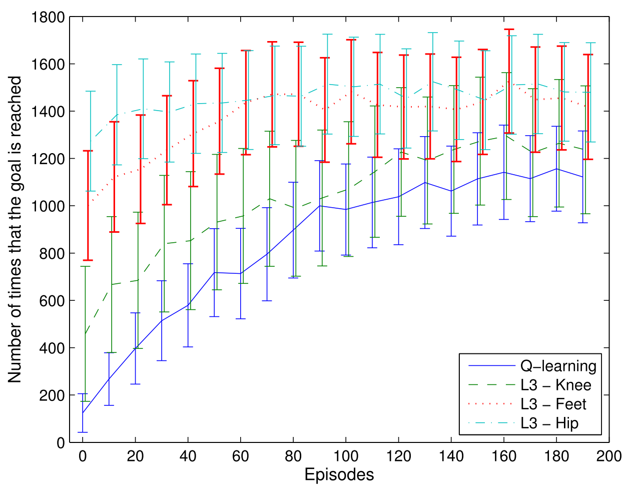
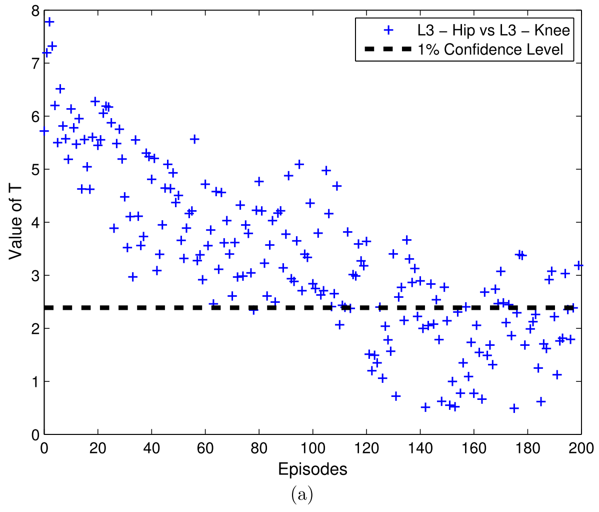


Fig. 10. The learning curves for Q-learning and L3-Q for three distinct joints of the humanoid robot.



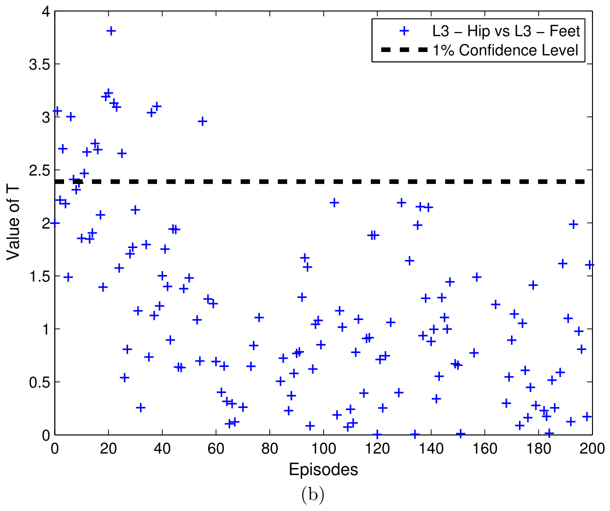


Fig. 11. Student's t-test between L3-Q used in the hip joint and in the knee joint (a), and between L3 used in the hip joint and in the feet joint (b).

6. Discussion

In the meta-algorithm L3, an element of the case base is used in the action selection rule to guide the search in the new domain, in the same way a heuristic is used in an informed search procedure. At the beginning of each learning episode, RL operates as a blind search method. As such, knowledge can be used to lift RL from a blind search style to an informed search procedure.

Experimental results (described in Section 5) were obtained for two instances of L3 (L3-Q and L3-SARSA(λ)) over two distinct domains, in order to confirm the generality of the method proposed. The experiments verified the initial claim that L3 outperforms its non transfer-learning versions, and also state-of-the-art transfer learning algorithms. The results show that L3 outperforms traditional RL algorithms, such as Q-Learning and SARSA(λ), HARL algorithms such as the HAQL and HA-SARSA(λ), and TL algorithms such as Taylor's MASTER [29] and TiMRLA Value-Addition algorithms [32]. It is worth pointing out that, in contrast to L3, HAQL and HA-SARSA(λ) presuppose user-defined domain knowledge. The fact that the L3 algorithms outperform other heuristically accelerated reinforcement learning (HARL) algorithms lets us conclude that learning the optimal policy of a similar task to be used as a heuristic in another domain is better than using a tailor-made heuristic based on expert knowledge.

L3 was compared with other state-of-the-art transfer learning algorithms: L3 was compared with the TiMRLA Value-Addition algorithm transfer learning algorithm [32] and Taylor's MASTER algorithm [29] on the Mountain Car experiment (Section 5.1). We did not find any competing algorithm in the literature to compare the Humanoid Robot Stabilisation experiment (presented in Section 5.2), which was a domain included in this paper to illustrate the generality of the algorithm proposed.

In this paper we compared L3 with HAQL but not with its case-base counterpart, CB-HAQL. In order to compare with CB-HAQL a tailor-made case base would have to be built. Not only this is a tedious process, but the results obtained would be very similar to those obtained with HAQL, since a tailor-made case base would be constructed using some pre-defined heuristics.

To the best of our knowledge, L3 is the only class of TL algorithms for Reinforcement Learning to date that uses the knowledge obtained in one domain as *heuristics* in another. This characteristic makes L3 robust to *negative* transfers: if the cases acquired in the source domain are not useful in the target domain, assuming them as heuristics will not speed up the learning procedure but, in the worst case (when every case in the case base is not applicable to the target domain), L3 will be as efficient as the original RL algorithm that it is based. In other words, if the case base contains no useful (or even misleading) information for the target domain, the agent is still able to learn the optimal policy for the domain using the RL component of the algorithm. As the value of the heuristic defined in Equation (4) is bounded, after a finite number of learning iterations, the correct value of the value function in that state will be learned.

The effect of negative transfer on L3 was presented in Fig. 6, that show the learning curves for L3 when it was given (as transferred knowledge from the source domain) a set with the N worst cases, obtained by random sampling the state set and selecting the worst action for that state.

Fig. 6 shows that L3 with negative transfer (curve L3-argmin) had the lowest jumpstart and could not learn anything until the 200th episode, after that it presented the best asymptotic performance and the best time to threshold, in contrast to the other cases analysed. This was due to the fact that the negative transfer delayed the algorithm initially (as expected), but after a number of learning iterations, the algorithm ignored the negative heuristics and avoided the states previously generated by these heuristics in later episodes. In other words, the learning curve of L3-argmin was steeper than the others because L3 learned to avoid bad states, from having to deal with the negative transference of knowledge in earlier episodes.

This verifies our hypothesis that L3 is robust to negative transfers due to its reliance on the transference of *heuristics* between domains. However, we were incapable of exactly bounding *the finite number of learning iterations* needed for L3 to recover from negative transfers. This is an open issue to be addressed in future work.

7. Conclusion and open issues

In this paper we investigated the performance of a new class of algorithms, called L3, which uses a case base to transfer knowledge as heuristics between domains. Two algorithms of this class were investigated: L3-Q, based on Q-Learning, and L3-SARSA(λ), based on the gradient descent SARSA(λ). The transference of heuristics across domains makes L3 robust to negative transfers, which is the major contribution of these algorithms to the state of the art of transfer learning in reinforcement learning. In order to show the generality of L3, this algorithm was applied to benchmark domains in RL whose results show that L3 outperforms Q-Learning (which is traditionally used as baseline algorithm), a Heuristically Accelerated RL algorithm and two state-of-the-art transfer learning algorithms.

However, L3 was not capable of mapping actions between domains in which there is a one-to-many possibility of mappings, and vice-versa. This issue shall be considered in future research. Future work shall also consider further possible relevance measures for case selection, other than the reward received. A possible candidate for such measure could be the probability assigned to a possible case to be used. In general, the more a case is used in the source domain, the more relevant it is. Whether or not the use of these cases in the target domain results in a better performance than that reported in this paper is an issued to be investigated.

In our experiments, cases are single step: they represent a single action to be taken in a state. However, we believe that this approach could easily be made more general, in the sense that sequences of actions can also be stored in the case base and used as heuristics. Experiments to validate this hypothesis are a matter for future work.

Acknowledgements

Luiz Celiberto Jr. and Reinaldo Bianchi acknowledge the support of FAPESP (grants 2012/14010-5 and 2011/19280-8). Paulo E. Santos acknowledges support from FAPESP (grant 2012/04089-3) and CNPq (grant PQ2 – 303331/2011-9). Ramon Lopez de Mantaras acknowledges the support of the Generalitat de Catalunya grant 2014-SGR-118.

References

[1] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[2] M.E. Taylor, P. Stone, Transfer learning for reinforcement learning domains: a survey, J. Mach. Learn. Res. 10 (1) (2009) 1633–1685.
[3] M.E. Taylor, P. Stone, An introduction to inter-task transfer for reinforcement learning, AI Mag. 32 (1) (2011) 15–34.
[4] C. Drummond, Accelerating reinforcement learning by composing solutions of automatically identified subtasks, J. Artif. Intell. Res. 16 (2002) 59–104.
[5] W.B. Knox, P. Stone, Combining manual feedback with subsequent MDP reward signals for reinforcement learning, in: Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems, AAMAS 2010, 2010, pp. 5–12.
[6] W.B. Knox, P. Stone, Reinforcement learning from simultaneous human and MDP reward, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 1, AAMAS '12, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, ISBN 0-9817381-1-7, 2012, pp. 475–482; ISBN 978-0-9817381-1-6.
[7] B. Price, C. Boutilier, Accelerating reinforcement learning through implicit imitation, J. Artif. Intell. Res. 19 (2003) 569–629.
[8] G. Konidaris, A. Barto, Autonomous shaping: knowledge transfer in reinforcement learning, in: Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 489–496.
[9] R.A.C. Bianchi, C.H.C. Ribeiro, A.H.R. Costa, Accelerating autonomous learning by using heuristic selection of actions, J. Heuristics (ISSN 1381-1231) 14 (2) (2008) 135–168.

[10] R.A.C. Bianchi, R. Ros, R. Lopez de Mantaras, Improving reinforcement learning by using case based heuristics, in: L. McGinty, D.C. Wilson (Eds.), Case-Based Reasoning Research and Development, 8th International Conference on Case-Based Reasoning, ICCBR 2009, Seattle, WA, USA, July 20–23, 2009, in: Lecture Notes in Computer Science, vol. 5650, Springer, ISBN 978-3-642-02997-4, 2009, pp. 75–89.
[11] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. (ISSN 1041-4347) 22 (10) (2010) 1345–1359.
[12] M. Klenk, D.W. Aha, M. Molineaux, The case for case-based transfer learning, AI Mag. 32 (1) (2011) 54–69.
[13] L.A. Celiberto Jr., J.P. Matsuura, R. Lopez de Mantaras, R.A.C. Bianchi, Using cases as heuristics in reinforcement learning: a transfer learning application, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, vol. 2, IJCAI'11, AAAI Press, 2011, pp. 1211–1217.
[14] S. Thrun, Explanation-Based Neural Network Learning: A Lifelong Learning Approach, Kluwer Academic Publishers, Boston, MA, 1996.
[15] A. Niculescu-Mizil, R. Caruana, Inductive transfer for Bayesian network structure learning, in: Unsupervised and Transfer Learning – Workshop Held at ICML 2011, Bellevue, Washington, USA, July 2, 2011, 2012, pp. 167–180.
[16] C. Lemke, M. Budka, B. Gabrys, Metalearning: a survey of trends and technologies, Artif. Intell. Rev. (ISSN 0269-2821) (2013) 1–14.
[17] R. Caruana, Multitask learning, Mach. Learn. (ISSN 0885-6125) 28 (1) (1997) 41–75.
[18] B.D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robot. Auton. Syst. 57 (5) (2009) 469–483.
[19] S. Griffith, K. Subramanian, J. Scholz, C.L. Isbell, A.L. Thomaz, Policy shaping: integrating human feedback with reinforcement learning, in: C.J.C. Burges, L. Bottou, Z. Ghahramani, K.Q. Weinberger (Eds.), NIPS, 2013, pp. 2625–2633.
[20] L. Saitta, J.-D. Zucker, Abstraction in Artificial Intelligence and Complex Systems, Springer Publishing Company, Incorporated, ISBN 9781461470519, 2013.
[21] A. von Hessling, A.K. Goel, Abstracting reusable cases from reinforcement learning, in: S. Brüninghaus (Ed.), 6th International Conference on Case-Based Reasoning, ICCBR 2005, Chicago, IL, USA, August 23–26, 2005, 2005, Workshop Proceedings.
[22] D.W. Aha, M. Molineaux, G. Sukthankar, Case-based reasoning in transfer learning, in: Proceedings of the 8th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development, ICCBR '09, Springer-Verlag, Berlin, Heidelberg, ISBN 978-3-642-02997-4, 2009, pp. 29–44.
[23] F. Fernández, M. Veloso, Probabilistic policy reuse in a reinforcement learning agent, in: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '06, ACM, New York, NY, USA, ISBN 1-59593-303-4, 2006, pp. 720–727.
[24] B. Banerjee, P. Stone, General game learning using knowledge transfer, in: The 20th International Joint Conference on Artificial Intelligence, 2007, pp. 672–677.
[25] V. Soni, S. Singh, Using homomorphisms to transfer options across continuous reinforcement learning domains, in: Proceedings of the 21st National Conference on Artificial Intelligence, vol. 1, AAAI Press, ISBN 978-1-57735-281-5, 2006, pp. 494–499.
[26] M.E. Taylor, N.K. Jong, P. Stone, Transferring instances for model-based reinforcement learning, in: Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Artificial Intelligence, vol. 5212, 2008, pp. 488–505.
[27] M. Snel, S. Whiteson, Multi-task reinforcement learning: shaping and feature selection, in: Proceedings of the 9th European Conference on Recent Advances in Reinforcement Learning, EWRL'11, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 237–248.
[28] H.B. Ammar, K. Tuyls, M.E. Taylor, K. Driessens, G. Weiss, Reinforcement learning transfer via sparse coding, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 1, AAMAS '12, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2012, pp. 383–390.
[29] M.E. Taylor, Autonomous inter-task transfer in reinforcement learning domains, Ph.D. thesis, University of Texas at Austin, Austin, TX, USA, 2008.
[30] H.B. Ammar, M.E. Taylor, Reinforcement learning transfer via common subspaces, in: Proceedings of the 11th International Conference on Adaptive and Learning Agents, ALA'11, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 21–36.
[31] A. Fachantidis, I. Partalas, M.E. Taylor, I. Vlahavas, Transfer learning via multiple inter-task mappings, in: Proceedings of the 9th European Conference on Recent Advances in Reinforcement Learning, EWRL'11, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 225–236.
[32] A. Fachantidis, I. Partalas, G. Tsoumakas, I. Vlahavas, Transferring task models in reinforcement learning agents, Neurocomputing (ISSN 0925-2312) 107 (2013) 23–32.
[33] J.S. Albus, A new approach to manipulator control: the Cerebellar Model Articulation Controller (CMAC), Trans. ASME G, J. Dyn. Syst. Meas. Control 97 (3) (1975) 220–227.
[34] M.L. Littman, C. Szepesvári, A generalized reinforcement learning model: convergence and applications, in: Proceedings of the 13th International Conference on Machine Learning, ICML'96, 1996, pp. 310–318.
[35] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, J. Artif. Intell. Res. 4 (1996) 237–285.
[36] C.J.C.H. Watkins, Learning from delayed rewards, Ph.D. thesis, University of Cambridge, 1989.
[37] G. Rummery, M. Niranjan, On-line Q-learning using connectionist systems, Technical report CUED/F-INFENG/TR 166, Cambridge University, Engineering Department, 1994.
[38] R.A.C. Bianchi, C.H.C. Ribeiro, A.H.R. Costa, Heuristically accelerated reinforcement learning: theoretical and experimental results, in: L.D. Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, P.J.F. Lucas (Eds.), 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, ECAI 2012, Montpellier, France, August 27–31, 2012, in: Front. Artif. Intell. Appl., vol. 242, IOS Press, ISBN 978-1-61499-097-0, 2012, pp. 169–174.
[39] R.A.C. Bianchi, C.H.C. Ribeiro, A.H.R. Costa, Heuristically accelerated Q-learning: a new approach to speed up reinforcement learning, Lect. Notes Artif. Intell. 3171 (2004) 245–254.
[40] L.A. Celiberto, C.H.C. Ribeiro, A.H.R. Costa, R.A.C. Bianchi, Heuristic reinforcement learning applied to RoboCup simulation agents, in: U. Visser, F. Ribeiro, T. Ohashi, F. Dellaert (Eds.), RoboCup, in: Lecture Notes in Computer Science, vol. 5001, Springer, ISBN 978-3-540-68846-4, 2007, pp. 220–227.
[41] J.A. Gurzoni, F. Tonidandel, R.A.C. Bianchi, Market-based dynamic task allocation using heuristically accelerated reinforcement learning, in: L. Antunes, H.S. Pinto (Eds.), EPIA, in: Lecture Notes in Computer Science, vol. 7026, Springer, ISBN 978-3-642-24768-2, 2011, pp. 365–376.
[42] R.A.C. Bianchi, C.H.C. Ribeiro, A.H.R. Costa, Heuristic selection of actions in multiagent reinforcement learning, in: M.M. Veloso (Ed.), IJCAI, 2007, pp. 690–695.
[43] R.A.C. Bianchi, R. Lopez de Mantaras, Case-based multiagent reinforcement learning: cases as heuristics for selection of actions, in: Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence, IOS Press, Amsterdam, The Netherlands, ISBN 978-1-60750-605-8, 2010, pp. 355–360.
[44] H.-D. Burkhard, Similarity and distance in case based reasoning, Fundam. Inform. (ISSN 0169-2968) 47 (3–4) (2001) 201–215.
[45] G.R. Finnie, Z. Sun, Similarity and metrics in case-based reasoning, Int. J. Intell. Syst. 17 (3) (2002) 273–287.
[46] R. Ros, J.L. Arcos, R. Lopez de Mantaras, M. Veloso, A case-based approach for coordinated action selection in robot soccer, Artif. Intell. (ISSN 0004-3702) 173 (9–10) (2009) 1014–1039.
[47] D.O. Hebb, The Organization of Behavior: A Neuropsychological Theory, Wiley, New York, ISBN 0-8058-4300-0, 1949.
[48] N. Doidge, The Brain That Changes Itself, Viking Press, 2007.
[49] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.
[50] M.R. Spiegel, Statistics, McGraw-Hill, 1998.
[51] U. Nehmzow, Scientific Methods in Mobile Robotics: Quantitative Analysis of Agent Behaviour, Springer-Verlag London Limited, London, ISBN 1846280192, 2006.

[52] A. Moore, Variable resolution dynamic programming: efficiently learning action maps in multivariate real-valued state-spaces, in: L. Birnbaum, G. Collins (Eds.), Machine Learning: Proceedings of the Eighth International Conference, Morgan Kaufmann, San Francisco, CA, 1991, pp. 333–337.
[53] M.E. Taylor, G. Kuhlmann, P. Stone, Autonomous transfer for reinforcement learning, in: L. Padgham, D.C. Parkes, J.P. Müller, S. Parsons (Eds.), 7th International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 1, AAMAS 2008, Estoril, Portugal, May 12–16, 2008, IFAAMAS, ISBN 978-0-9817381-0-9, 2008, pp. 283–290.
[54] R.S. Sutton, Mountain car software, http://webdocs.cs.ualberta.ca/~sutton/MountainCar/MountainCar.html, 2000, last accessed 20/11/2014.
[55] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. (ISSN 1049-3301) 8 (1) (1998) 3–30.
[56] I. Partalas, Mountain car 3D CPP in RL library, http://library.rl-community.org/wiki/Mountain_Car_3D_(CPP), last accessed 20/11/2014, 2014.
[57] I. Partalas, The value-addition algorithm for mountain car, http://mlkd.csd.auth.gr/transfer-learning.html, last accessed 20/11/2014, 2014.
[58] R.A.C. Bianchi, M. Martins, C.H.C. Ribeiro, A.H.R. Costa, Heuristically-accelerated multiagent reinforcement learning, cybernetics, IEEE Trans. Cybern. (ISSN 2168-2267) 44 (2) (2014) 252–265.
[59] B. Tanner, A. White, RL-glue: language-independent software for reinforcement-learning experiments, J. Mach. Learn. Res. 10 (2009) 2133–2136.
[60] J. Boedecker, K. Dorer, M. Rollmann, Y. Xu, F. Xue, M. Buchta, H. Vatankhah, SimSpark Users's Manual, RoboCup Federation, 2010.