Using Cases as Heuristics in Reinforcement Learning: a Transfer Learning Application

Luiz A. Celiberto Jr.
Dept. of Electrical Engineering, Technological Institute of Aeronautics (ITA)
São José dos Campos, Brazil
celibertojr@fei.edu.br

Jackson P. Matsuura
Dept. of Electrical Engineering, Technological Institute of Aeronautics (ITA)
São José dos Campos, Brazil
jackson@ita.br

Ramon Lopez de Mantaras
Artificial Intelligence Research Institute (IIIA-CSIC)
Campus Universitat Autonoma de Barcelona, Bellaterra, Spain
mantaras@iiia.csic.es

Reinaldo A. C. Bianchi
Dept. of Electrical Engineering, Centro Universitário da FEI
São Bernardo do Campo, Brazil
rbianchi@fei.edu.br

Abstract

In this paper we propose to combine three AI techniques to speed up a Reinforcement Learning algorithm in a Transfer Learning problem: Case-based Reasoning, Heuristically Accelerated Reinforcement Learning and Neural Networks. To do so, we propose a new algorithm, called L3, which works in 3 stages: in the first stage, it uses Reinforcement Learning to learn how to perform one task, and stores the optimal policy for this problem as a case-base; in the second stage, it uses a Neural Network to map actions from one domain to actions in the other domain and; in the third stage, it uses the case-base learned in the first stage as heuristics to speed up the learning performance in a related, but different, task. The RL algorithm used in the first phase is the Q-learning and in the third phase is the recently proposed Case-based Heuristically Accelerated Q-learning. A set of empirical evaluations were conducted in transferring the learning between two domains, the Acrobot and the Robocup 3D: the policy learned during the solution of the Acrobot Problem is transferred and used to speed up the learning of stability policies for a humanoid robot in the Robocup 3D simulator. The results show that the use of this algorithm can lead to a significant improvement in the performance of the agent.

1 Introduction

One of the main problems of Reinforcement Learning (RL) [Sutton and Barto, 1998] algorithms is that they typically suffers from very slow learning rates, requiring a huge number of iterations to converge on a good solution. This problem becomes worse in tasks with high dimensional or continuous state spaces and when the system is given sparse rewards.

One way to speed up RL algorithms is by making use of a conveniently chosen heuristic function, which is used for selecting appropriate actions to perform in order to guide exploration during the learning process [Bianchi *et al.*, 2008]. Several methods have been successfully applied for defining the heuristic function, including the reuse of previously learned policies, using a Case-Based Reasoning approach [Bianchi *et al.*, 2009]. Another way to speed up a RL algorithm is by using Transfer Learning, a paradigm of machine learning that reuses knowledge accumulated in a previous task to speed up the learning of a novel, but related, target task [Taylor and Stone, 2009].

This paper investigates the use of the Case-Based Heuristically Accelerated Reinforcement Learning (CB-HARL) algorithm [Bianchi *et al.*, 2009] as a means to transfer learning acquired by one agent during its training in one problem to another agent that has to learn how to solve a similar, but more complex, problem. To do so, we propose a new algorithm, called L3, which works in 3 stages: in the first stage, it uses the Q-learning algorithm [Watkins, 1989] to learn how to perform one task, and stores the optimal policy for this problem as a case-base; in the second stage, it uses a Neural Network to map actions from one domain to actions in the other domain and; in the third stage, it uses the case-base learned in the first stage as heuristics in the CB-HARL algorithm, speeding up the learning process.

Experiments in this work were conducted in two domains: the Acrobot [Sutton and Barto, 1998], where the actions learned during the solution of the problem are stored as a case-base; and the Robocup 3D Simulator, where the case based is used to speed up the learning of stability policies for a humanoid robot. Nevertheless, the technique described in this work is domain independent and can be used to solve a wide range of problems.

The paper is organized as follows: Section 2 describes the Case Based Reasoning technique and Section 3 briefly reviews the heuristic approach to speed up RL and the CB-HAQL algorithm. Section 4 describes the Transfer Learning problem and Section 5 describes the combination of the techniques and the L3 algorithm. Section 6 describes the experiments and results and finally, Section 7 concludes this work.

2 Case Based Reasoning

Case Based Reasoning [de Mántaras *et al.*, 2005] is an AI technique that has been shown to be useful in a multitude of domains. CBR uses knowledge of previous situations (cases) to solve new problems, by finding a similar past case and reusing it in the new problem situation. In the CBR approach, a case usually describes a problem and its solution, i.e., the state of the world in a given instant and action to perform to solve that problem.

According to López de Mántaras *et al* [2005], solving a problem by CBR involves "obtaining a problem description, measuring the similarity of the current problem to previous problems stored in a case base with their known solutions, retrieving one or more similar cases, and attempting to reuse the solution of the retrieved case(s), possibly after adapting it to account for differences in problem descriptions". Other steps that are usually found in CBR systems are the evaluation of the proposed solution, the revision of the solution, if required in light of its evaluation, and the retention (learning) of a new case, if the system has learned to solve a new problem.

In general, in CBR a case is composed of a problem description (P) and the corresponding description of the solution (A). Therefore, the case definition is formally described as a tuple:

$$case = (P, A).$$

The case retrieval process consists in obtaining from the base the most similar case, the retrieved case. Therefore, it is necessary to compute the similarity between the current problem and the cases in the base. The similarity function indicates how similar a problem and a case are. In this work this function is defined by the quadratic distance between the problem and the case.

3 Combining CBR and Heuristics in Reinforcement Learning

The RL problem can be formulated as a discrete time, finite state, finite action Markov Decision Process (MDP). The learning environment can be modeled by a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where: $\mathcal{S}$: is a finite set of states. $\mathcal{A}$: is a finite set of actions that the agent can perform. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$: is a state transition function, where $\Pi(\mathcal{S})$ is a probability distribution over $\mathcal{S}$. $T(s, a, s')$ represents the probability of moving from state s to s' by performing action a. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$: is a scalar reward function.

A Heuristically Accelerated Reinforcement Learning (HARL) algorithm [Bianchi *et al.*, 2008] is a way to solve a MDP problem with explicit use of a heuristic function $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ for influencing the choice of actions by the learning agent. $H(s, a)$ defines the heuristic that indicates the importance of performing action a when visiting state s. The heuristic function is strongly associated with the policy indicating which action must be taken regardless of the action-value of the other actions that could be used in the state.

The first HARL algorithm proposed was the Heuristically Accelerated Q–Learning (HAQL) [Bianchi *et al.*, 2004], as an extension of the Q–Learning algorithm [Watkins, 1989]. The only difference between the two algorithms is that in the HAQL makes use of an heuristic function $H(s, a)$ in the $\epsilon - greedy$ action choice rule, that can be written as:

$$\pi(s) = \begin{cases} \arg\max_a \left[\hat{Q}(s, a) + \xi H(s, a)^\beta \right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise,} \end{cases} \quad (1)$$

where $H(s, a)$ is the heuristic function that plays a role in the action choice, ξ and β are design parameters that control the influence of the heuristic function, q is a random value with uniform probability in [0,1] and p ($0 \leq p \leq 1$) is the parameter which defines the exploration/exploitation trade-off and a_{random} is an action randomly chosen among those available in state s.

Table 1: The CB-HAQL algorithm [Bianchi *et al.*, 2009].

Initialize $\hat{Q}_t(s, a)$ and $H_t(s, a)$ arbitrarily.
Repeat (for each episode):
  Initialize s.
  Repeat (for each step):
    Compute similarity and cost.
    If there is a case that can be reused:
      Retrieve and Adapt if necessary.
      Compute $H_t(s, a)$ using Equation 2 with the actions suggested by the case selected.
    Select an action a using equation 1.
    Execute the action a, observe $r(s, a)$, s'.
    Update the values of $Q(s, a)$.
    $s \leftarrow s'$.
  Until s is terminal.
Until some stopping criterion is reached.

The Q-values are updated using the traditional Q-learning equation. As a general rule, the value of $H(s, a)$ used in HAQL should be higher than the variation among the $\hat{Q}(s, a)$ values for the same $s \in \mathcal{S}$, in such a way that it can influence the choice of actions, and it should be as low as possible in order to minimize the error. It can be defined as:

$$H(s, a) = \begin{cases} \max_i \hat{Q}(s, i) - \hat{Q}(s, a) + \eta \text{ if } a = \pi^H(s), \\ 0 \text{ otherwise.} \end{cases} \quad (2)$$

where η is a small real value (usually 1) and $\pi^H(s)$ is the action suggested by the heuristic policy.

In order to provide HARL algorithms the capability of reusing previous knowledge from a domain, Bianchi *et al.* [2009] proposed the Case Based HAQL, which extends the HAQL algorithm with the abilities to retrieve a case stored in a base, adapt it to the current situation, and build a heuristic function that corresponds to the case.

In this new algorithm steps were added before the action selection is made to compute the similarity of the cases with the current state and the cost of adaptation of these cases. A case is retrieved if the similarity is above a certain threshold, and the adaptation cost is low. After a case is retrieved, an heuristic is computed using Equation 2 and the sequence of actions suggested by the case selected. This heuristic is used for a certain amount of time, equal to the number of actions of the retrieved case. After that time, a new case can be retrieved. The complete CB-HAQL algorithm is presented in Table 1.

Several authors have been studying the use of CBR together with RL: Sharma *et al* [2007] makes use of CBR as a function approximator for RL, and RL as revision algorithm for CBR in a hybrid architecture system; Gabel and Riedmiller [2005] also makes use of CBR in the task of approximating a function over high-dimensional, continuous spaces; Juell and Paulson [2003] exploit the use of RL to learn similarity metrics in response to feedback from the environment; Auslander *et al* [2008] uses CBR to adapt quickly an RL

agent to changing conditions of the environment by the use of previously stored policies and Li, Zonghai and Feng [2002] proposes an algorithm that makes use of knowledge acquired by reinforcement learning to construct and extend a case base.

4 Transfer Learning

According to Taylor and Stone [2009], only recently the use of Transfer Learning for Reinforcement Learning has gained attention in the artificial intelligence community. Transfer Learning is not a new idea: it has been studied in the psychological literature on transfer of learning since the work of Thorndike and Woodworth [1901]. Also, TL has been used to transfer between machine learning tasks for some time now. These works usually study transfer of learning in the context of classification, multitask learning and inductive learning.

Is possible to divided the TL in two main categories: intra-domain transfer, where TL is used to solve a new task within a given domain and cross-domain transfer where transfer is made between domains. Usually, Intra-domain transfer uses the same space state and transforms primitive actions in more complex actions to be uses in new tasks within this domain. Cross-domain transfer tries to find similar structure between the source and target task to transfer the learning. An RL agent must, at least, perform the following steps [Taylor and Stone, 2009]:

- Given a target task, select an appropriate source task or set of tasks from which to transfer.
- Learn how the source task(s) and target task are related.
- Effectively transfer knowledge from the source task(s) to the target task.

Transfer Learning is a very important tool to speed up RL algorithms because, in RL, even a small change on the configuration of a problem may requires a complete new training. With TL, what an agent has learned can be transferred to a new situation, helping it to learn faster. Drummond [2002] was probably the first to use CBR to speed up RL, proposing to accelerate RL by transferring parts of previously learned solutions to a new problem, exploiting the results of prior learning to speed up the process. More recent works on TL that have combined CBR and RL include, for example, van Hessing and Goel [2005] proposal of a technique for abstracting reusable cases from RL, enabling the transfer of acquired knowledge to other instances of the same problem and Aha *et al.* [2009] method for recognizing intent in a source task, and then applying that knowledge to improve the performance of a case-based reinforcement learner in a target task.

Other important works that focus on other aspects of the use of TL in TL include: Fernandez and Veloso [2006], that focus on exploration and policy reuse; Torrey *et al.* [2005], which create rules from the learned policy to help another agent. Banerjee and Stone [2007] transfer knowledge learned in one game to expedite learning in many other games; Soni and Singh [2006] used homomorphisms to take a policy from one MDP and transfer it to the other, directly; Taylor *et al* [2008] shows that an action and state variable mapping can be learned, but does not directly transfers the policy between domains, using instances from the source task as a previously observed transition in the target task; and Lazaric *et al.* [2008] transfer parts of the space states to help to accelerate learning. Finally, Taylor and Stone [2009] present a framework that classifies transfer learning methods and survey the existing literature.

Table 2: The L3 algorithm.

1.a)	Use the Q-learning algorithm to compute the optimal policy for the source domain.
1.b)	Create a case-base.
2)	Map actions from source domain to target domain using a Neural Network.
3)	Use the case base in the CB-HAQL algorithm to solve the problem in the target domain.

5 The L3 algorithm

To transfer the cases between learning agents in two domains we propose the L3 algorithm, which works in 3 stages: it learns how to perform a task in the source domain, and stores the optimal policy for this problem as a case-base; it maps actions from the source domain to actions in the target domain; and it uses the case-base learned in the first stage as heuristics in the CB-HARL algorithm (see Table 2). The name of the algorithm comes from the fact that it learns three times (the source task, the mapping and the target task).

The main motivation of using cases as heuristics to transfer the learning is that the heuristic function is an action policy modifier which does not interfere with the standard bootstrap-like update mechanism of RL the algorithm: the new L3 algorithm differs from the Q-learning only in the way exploration is carried out, which allows many theoretical conclusions obtained for the Q-learning (such as Convergence Theorem) to remain valid for the L3. Other proposals combining CBR and RL cannot guarantee this.

5.1 Stage 1: Learning the source task and construction of the Case-base

In the first stage, the case base construction, the Q-learning algorithm is used to compute the optimal policy for the source domain. After the learning stabilizes a case based is built from the learned policy, with a pre-defined number of cases.

Similar to the model proposed by Ros [Ros *et al.*, 2009], 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 state $s \in \mathcal{S}$); A is an action (or a sequence of actions) that must be performed to solve the problem and; R is the expected return for performing the action, which indicates the quality of the action stored in this case.

5.2 Stage 2: Mapping of action between domains

In the second stage, a simple Neural Network learns how to relate the actions between the source task and the target task (the mapping between the states is assumed). In this network, the input nodes correspond to the set of possible actions in the target domain, and the output nodes corresponds to the set of

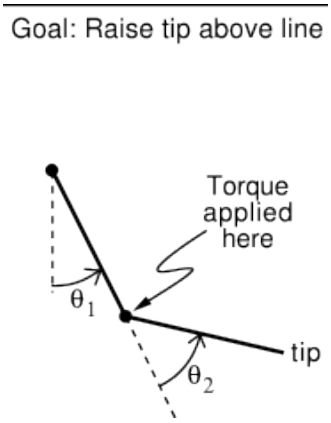

Figure 1: The Acrobot (from Sutton and Barto, 1998)

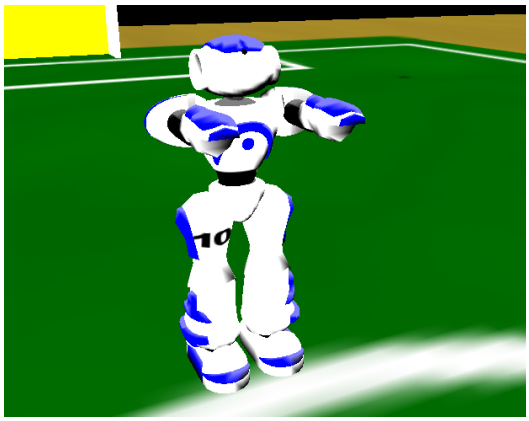

Figure 2: The Nao Robot in 3D (from Boedecker *et al.*, 2010)

actions of the source domain (usually, the source domain has fewer actions than the target domain).

To learn the weights of the network, a set of random actions is executed in both simulators, and the results of the actions are observed (the result is the distribution over the next states). If the results of the two actions are similar (for example, both actions lead to an increase in the x position of a robot), the weight that links both actions is increased to strengthen the correlation between the actions, and the weights of the other connections are decreased; if the results are not similar, or it is not possible to observe any similarity, the weights are left as they are.

This scheme can be formalized as a linear, single layer, forward feed linear perceptron using the Hebb Learning rule, where the input and output vectors should be in the bipolar form (1 or -1) and the activation function of the output nodes are binary. The weights are initially zero and they are adjusted each time the results of two actions are considered similar (actions with similar results are the input vector for training), using the following formula:

$$\Delta w_i = \eta x_i y \tag{3}$$

where w_i is weight for input i; x_i is the input value i (the action in the target domain); y is the output value (the action in the source domain) and η is learning rate. The result of this training is a table that describes the relation between the actions in both domains.

5.3 Stage 3: Reusing the case base in the CB-HARL algorithm

In the last stage, the previously stored case base is used in the CB-HAQL algorithm to speed up the learning of the task in the target domain.

Case retrieval is in general driven by a similarity measure between the new problem and the solved problems in the case base. In this work we use the case retrieval method proposed by Ros *et al.* (2009), which considers the similarity between the problem and the case (the similarity is computed using a Gaussian distance between the case and the problem), the cost of adapting the problem to the case, and the applicability of the solution of the case. The cost of adapting the problem to the case is computed as a function of the distances between the features in the problem and the ones specified in the case. The complete case retrieval algorithm is described in detail in Ros *et al.* (2009).

After a case is retrieved, a heuristic is computed using Equation 2 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, the CB-HAQL algorithm will behave as the traditional Q-learning algorithm.

Our approach differs from previous research combining CBR and RL because the policy learned in one domain is stored as a case base and then used in a new domain as a heuristic: it 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 method. At the beginning of each learning episode, RL operates as a blind search method. However, cases can be used to improve RL from a blind search method to an informed search one. By doing this, if the case base contains a case that can be used in a given situation, then there will be a speed up in the convergence time. But if the case base does not contain any useful case – or even if it contains cases that implement wrong solutions to the problem – the agent will still learn the optimal solution by using the RL component of the algorithm.

6 The Transfer Learning Experience

In the this section we present an application of the L3 algorithm, where cases acquired in the Acrobot domain are used to speed up the learning of stability of a humanoid robot in the Robocup 3D Soccer Simulator domain.

The Acrobot [Sutton and Barto, 1998] (Figure 1) is a two-link, underactuated robot where the first joint cannot exert torque, but the second joint can. This system has four continuous state variables: two joint positions, θ_1 and θ_2, and two joint velocities $\dot{\theta}_1$ and $\dot{\theta}_2$. The goal is to swing the end-point above the bar by an amount equal to equilibrium position ($\theta_1 = (\pi/2), \theta_2 = 0$), starting from the initial state $\theta_1 = \theta_2 = 0$. There are three possible actions: positive torque, negative torque, and no torque [Sutton and Barto, 1998].

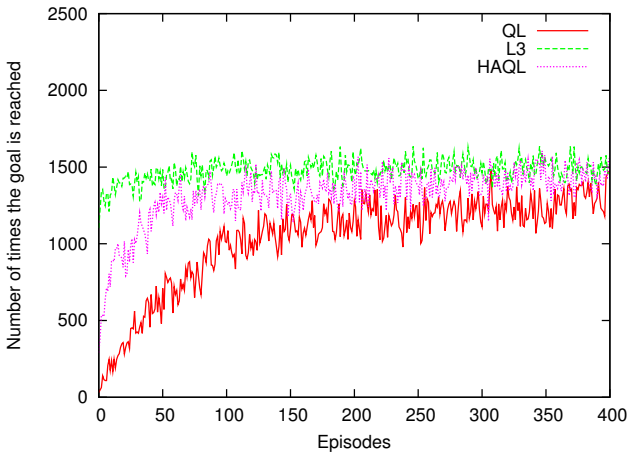

Figure 3: The learning curves for the Q–learning, HAQL and L3 algorithms

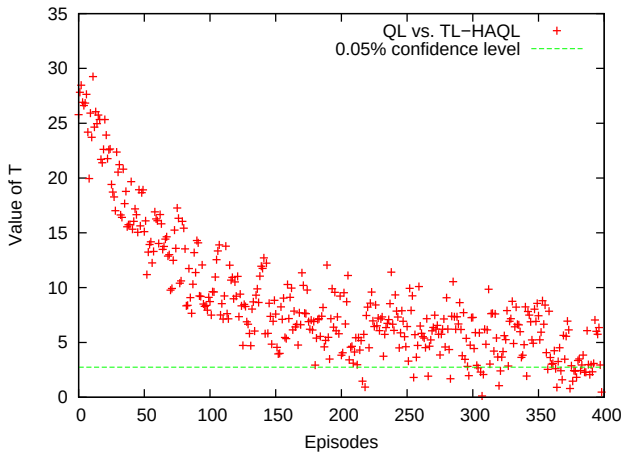

Figure 4: Student's t-Test between Q-learning and L3.

Table 3: Action-Mapping.

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

The RoboCup 3D Simulated Soccer League aim is to help the RoboCup federation to achieve it's goal of developing a team of fully autonomous humanoid robots that can win against the human world soccer champion team in 2050, by developing a realistic simulator that allows agents to control humanoid robots competing against one another. The current robot model used by the simulator is based on the Nao Robot (Figure 2) by Aldebaran Robotics. This robot is a biped humanoid with 22 degrees of freedom, with height about 57cm and weight around 4.5kg [Boedecker *et al.*, 2010]. The Nao robot is equipped with various sensors and effectors, some of them reproduced in the simulator, for example: angle sensors in each joint, a gyroscope, an accelerometer and a force sensor that provides information about the force applied upon the sole of each foot.

In the experiment of learning a stability policy for this robot, we controlled only three of the robot's joints to help to find the equilibrium position: Hip Pitch, Knee Pitch, Foot Pitch (with left and right joint having the same position). All the other joints of the robot are kept in the same position. At each time step, the robot can use one of seven actions possible: $+0.5°$ Hip Pitch, $+0.5°$ Knee Pitch, $+0.5°$ Foot Pitch, $-0.5°$ Hip Pitch, $-0.5°$ Knee Pitch, $-0.5°$ Foot Pitch and 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).

The first stage of the L3 algorithm is to build the case-base to be used by the CB-HAQL. To do so, the Q–learning algorithm is used in the Acrobot domain for 10,000 episodes (each episode ends either after 20,000 steps or when the agent find the goal state). Acquiring cases begins when the learning stabilizes ($\hat{Q}(s', a') - \hat{Q}(s, a) \sim 0$) which happens near the 9,000th episode. From the episode 9,000 and beyond, 500 cases are acquired by sampling the action-state set randomly. During this sampling, if a case contains the worst action for that state (i.e., the one with the lowest Q value), this case is discarded. Other possible ways to build the case base were tested (using the 500 actions with the best R, for example) but, surprisingly, the approach used in this work was the one that produced the best results. In this experiment, cases are single step: they represent a single action to be taken in a state (but the approach is general – sequences of actions can also be stored in the case base).

For the second stage of the L3 algorithm, it is needed to relate the actions between the two domains. To compare the output of the actions, we compared the Acrobot's $\theta 1$ to the movement of the Nao Robot ankle (foot pitch) and $\theta 2$ with the movement of it's knee. All other joint and velocities were left out of this comparison, which can be considered as an intra-domain transfer of learning.

In the experiment, the neural network input consists of the seven action used in the Robocup 3D Simulator and the outputs are the three possible actions in the Acrobot. Table 3 shows the result of the automatic action-mapping.

At the last stage of the L3 algorithm, the case-base is used in the CB-HAQL algorithm to learn the Nao Robot's equilibrium position. The features used to compute the distance between a case and the problem is the angles in the joints (the states in both domains).

To verify the hypothesis that the L3 algorithm improves the learning rate of the system, we compare the obtained results with two other algorithms: the Q-learning and the HAQL. Thirty training sessions were executed for these three algorithms, each session consisting of 400 episodes, each episode having 120 seconds of duration. The results of the three algorithms can be seen in Figure 3. This Figure shows the leaning curves (i.e., how many times per episode the agent reached the goal).

It can be seen that the performance of the Q–learning is worse than that of the HAQL, which is worse than that of

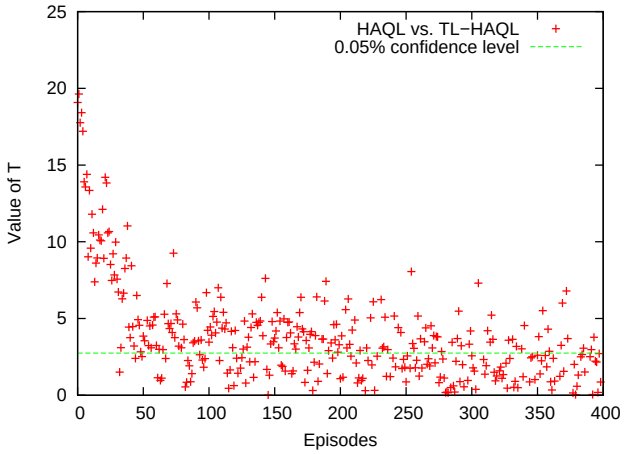

Figure 5: Student's t-Test between HAQL and L3.

Table 4: Learning time for the three algorithms.

Algorithm	Time (in minutes)
Q-learning	700
HAQL	200
L3	90

the L3 at the initial learning phase; later the performance of the three algorithms becomes more similar, as expected. It is important to remember that the HAQL uses a domain knowledge that must be introduced by the programmer, while the L3 uses less domain knowledge.

Student's t-Test was used to verify the hypothesis that the transfer of learning speeds up the learning process. For the experiments the absolute value of T was computed for each episode using the same data presented in Figure 3. The greater the absolute value of T, the more significantly different is the result. The dotted line indicates the 0.05% confidence limit, i.e. results above the line are different and the probability for this statement to be erroneous is 0.05%. The results, presented in Figures 5 and 4, show that L3 performs significantly better than Q–learning until the 350^{th} episode and HAQL until the 50^{th} episode, with a level of confidence greater than 99.95%. After that, the results became closer.

Finally, the learning time of the L3 algorithm was compared with that of the other algorithms (Table 4). The 3D simulator takes 120 seconds for each episode, regardless of the algorithm it is running. The Q-learning takes 350 episodes (700 minutes) to reach optimality and the HAQL takes 100 episodes (200 minutes). The L3 takes 40 episodes to converge. Adding this to the learning time for the Acrobot (approx. 10 minutes) and the time to build the case-base (less than 1 minute), gives a total time of approximately 90 minutes for L3. The time to learn the action mapping was not taken into account when computing the total time of the L3 algorithm because it was much smaller than the rest of the learning times: very few data is used in the this step – in this experiment, it ranged from 50 to 100 steps – much less than one complete episode.

The parameters used in all the experiments were the same: $\alpha = 0.25$, $\gamma = 0.9$, the exploration/ exploitation rate $= 0.1$ and the Q table initialized with zeroes. HAQL and the L3 algorithms use $\eta = 1$. The reward is -1 on all steps, except when the goal is reached. In this case the reward is +1. The heuristic used in the HAQL algorithm was defined using a simple rule: if the Robocup 3D is laying forward, move the HIP back ($-0.5°$), if it is laying backwards, move the HIP forward ($+0.5°$). The algorithm uses a discretized Q-table, composed by the angles of the joints, discretized at one-degree intervals.

7 Conclusion

In this paper we proposed a new algorithm, called L3, which combines three AI techniques to speed up a Reinforcement Learning algorithm in a Transfer Learning problem: Case-based Reasoning, Heuristically Accelerated Reinforcement Learning and Neural Networks.

The experiments showed that transferring the policy learned by an agent in one domain to another agent in a different domain by means of the case-base speeds up the convergence time of the L3 algorithm, when compared to the Q–learning or the HAQL algorithm. Although this work used the Q-learning algorithm in the L3 algorithm, the proposed method can be integrated with any TD learning algorithm.

The major problem with this approach is to define when two actions have a similar result. In this work, which we considered as an intra-domain transfer of learning, the variables which should be compared were defined by the programmer. A solution to this problem, which is well known in the TL literature, is the most important work to be done in the future. Another issue with the algorithm proposed here is that the neural network used is very simple - other network architectures may lead to better results.

Acknowledgments

Luiz Celiberto Jr. acknowledges the support of CAPES and Reinaldo Bianchi acknowledges the support of the CNPq (201591/2007-3) and FAPESP (2011/07127-0). This work has been partially funded by the AGAUR 2009-SGR-1434 grant of the Generalitat de Catalunya and the NEXT-CBR MICINN TIN2009-13692 project.

References

[Aha *et al.*, 2009] David W. Aha, Matthew Molineaux, and Gita Sukthankar. Case-based reasoning in transfer learning. In Lorraine McGinty and David C. Wilson, editors, *8th International Conference on Case-Based Reasoning*, volume 5650 of *Lecture Notes in Computer Science*, pages 29–44. Springer, 2009.

[Auslander *et al.*, 2008] Bryan Auslander, Stephen Lee-Urban, Chad Hogg, and Héctor Muñoz-Avila. Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. In Klaus-Dieter Althoff, Ralph Bergmann, Mirjam Minor, and Alexandre Hanft, editors, *9th European Conference on Case-Based Reasoning*, volume 5239 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2008.

[Banerjee and Stone, 2007] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 672–677. AAAI Press, 2007.

[Bianchi *et al.*, 2004] Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, and Anna H. R. Costa. Heuristically Accelerated Q-learning: a new approach to speed up reinforcement learning. *Lecture Notes in Artificial Intelligence*, 3171:245–254, 2004.

[Bianchi *et al.*, 2008] Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, and Anna H. R. Costa. Accelerating autonomous learning by using heuristic selection of actions. *Journal of Heuristics*, 14(2):135–168, 2008.

[Bianchi *et al.*, 2009] Reinaldo A. C. Bianchi, Raquel Ros, and Ramon López de Mántaras. Improving reinforcement learning by using case based heuristics. In Lorraine McGinty and David C. Wilson, editors, *8th International Conference on Case-Based Reasoning*, volume 5650 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 2009.

[Boedecker *et al.*, 2010] Joschka Boedecker, Klaus Dorer, Markus Rollmann andYuan Xu, Feng Xue, Marian Buchta, and Hedayat Vatankhah. Spark 3D Simulation System. 2010.

[de Mántaras *et al.*, 2005] Ramon López de Mántaras, David McSherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, Mary Lou Maher, Michael T. Cox, Kenneth Forbus, Mark Keane, Agnar Aamodt, and Ian Watson. Retrieval, reuse, revision and retention in case-based reasoning. *Knowl. Eng. Rev.*, 20(3):215–240, 2005.

[Drummond, 2002] Chris Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, 16:59–104, 2002.

[Fernández and Veloso, 2006] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss and Peter Stone, editors, *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 720–727. ACM, 2006.

[Gabel and Riedmiller, 2005] Thomas Gabel and Martin A. Riedmiller. CBR for state value function approximation in reinforcement learning. In Héctor Muñoz-Avila and Francesco Ricci, editors, *6th International Conference on Case-Based Reasoning*, volume 3620 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2005.

[Juell and Paulson, 2003] Paul Juell and Patrick Paulson. Using reinforcement learning for similarity assessment in case-based systems. *IEEE Intelligent Systems*, 18(4):60–67, 2003.

[Lazaric *et al.*, 2008] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In William W. Cohen, Andrew McCallum and Sam T. Roweis, editors, *25th International Conference on Machine Learning*, pages 544–551. ACM, 2008.

[Li *et al.*, 2002] Yang Li, Chen Zonghai, and Chen Feng. A case-based reinforcement learning for probe robot path planning. In *4th World Congress on Intelligent Control and Automation*, pages 1161– 1165. 2002.

[Ros *et al.*, 2009] Raquel Ros, Josep Lluis Arcos, Ramon López de Mántaras, and Manuela Veloso. A case-based approach for coordinated action selection in robot soccer. *Artificial Intelligence*, 173(9-10):1014–1039, 2009.

[Sharma *et al.*, 2007] Manu Sharma, Michael Holmes, Juan Carlos Santamaría, Arya Irani, Charles Lee Isbell Jr., and Ashwin Ram. Transfer learning in real-time strategy games using hybrid CBR/RL. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1041–1046. AAAI Press, 2007.

[Soni and Singh, 2006] Vishal Soni and Satinder Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the 21st National Conference on Artificial Intelligence*, volume 1, pages 494–499. AAAI Press, 2006.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[Taylor and Stone, 2009] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[Taylor *et al.*, 2008] Matthew E. Taylor, Nicholas K. Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In Walter Daelemans, Bart Goethals and Katharina Morik, editors, *19th European Conference on Machine Learning*, volume 5212 of *Lecture Notes in Artificial Intelligence*, pages 488–505. Springer, 2008.

[Thorndike and Woodworth, 1901] E. L. Thorndike and R. S. Woodworth. The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review*, 8:247–261, 1901.

[Torrey *et al.*, 2005] Lisa Torrey, Trevor Walker, Jude W. Shavlik, and Richard Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In João Gama, Rui Camacho, Alípio Jorge, and Luís Torgo, editors, *16th European Conference on Machine Learning*, volume 3720 of *Lecture Notes in Computer Science*, pages 412–424. Springer, 2005.

[von Hessling and Goel, 2005] Andreas von Hessling and Ashok K. Goel. Abstracting reusable cases from reinforcement learning. In Stefanie Brüninghaus, editor, *6th International Conference on Case-Based Reasoning, Workshop Proceedings*, pages 227–236, 2005.

[Watkins, 1989] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.