Heuristic Reinforcement Learning applied to RoboCup Simulation Agents

Luiz A. Celiberto Jr.[1,2], Carlos H. C. Ribeiro[2], Anna Helena Reali Costa[3], and Reinaldo A. C. Bianchi[1]

[1] Centro Universitário da FEI
Av. Humberto de Alencar Castelo Branco, 3972.
09850-901 – São Bernardo do Campo – SP, Brazil.

[2] Instituto Tecnológico de Aeronáutica
Praça Mal. Eduardo Gomes, 50.
12228-900 – São José dos Campos – SP, Brazil.

[3] Laboratório de Técnicas Inteligentes
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav. 3, 158. 05508-900 – São Paulo – SP, Brazil.

celibertojr@uol.com.br, carlos@ita.br, anna.reali@poli.usp.br, rbianchi@fei.edu.br

Abstract. This paper describes the design and implementation of robotic agents for the RoboCup Simulation 2D category that learns using a recently proposed Heuristic Reinforcement Learning algorithm, the Heuristically Accelerated Q–Learning (HAQL). This algorithm allows the use of heuristics to speed up the well-known Reinforcement Learning algorithm Q–Learning. A heuristic function that influences the choice of the actions characterizes the HAQL algorithm. A set of empirical evaluations was conducted in the RoboCup 2D Simulator, and experimental results show that even very simple heuristics enhances significantly the performance of the agents.

Keywords: Reinforcement Learning, Cognitive Robotics, RoboCup Simulation 2D.

1 Introduction

Reinforcement Learning (RL) techniques have been attracting a great deal of attention in the context of multiagent robotic systems. The reasons frequently cited for such attractiveness are: the existence of strong theoretical guarantees on convergence [9], they are easy to use, and they provide model-free learning of adequate control strategies. Besides that, they also have been successfully applied to solve a wide variety of control and planning problems.

However, one of the main problems with RL algorithms is that they typically suffer from very slow learning rates, requiring a huge number of iterations to converge to 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 of the reasons for the slow learning rates is that most RL algorithms assumes that neither an analytical model nor a sampling model of the problem is available *a priori*. However, in some cases, there is domain knowledge that could be used to speed up the learning process.

As a way to add domain knowledge to help in the solution of the RL problem, a recently proposed Heuristic Reinforcement Learning algorithm – the Heuristically Accelerated Q–Learning (HAQL) [1] – uses a heuristic function that influences the choice of the action to speed up the well-known RL algorithm Q–Learning. This paper investigates the use of HAQL to speed up the learning process of teams of mobile autonomous robotic agents acting in a concurrent multiagent environment, the RoboCup 2D Simulator. It is organized as follows: section 2 describes the Q–learning algorithm. Section 3 describes the HAQL and its formalization using a heuristic function. Section 4 describes the robotic soccer domain used in the experiments, presents the experiments performed, and shows the results obtained. Finally, Section 5 summarizes some important points learned from this research and outlines future work.

2 Reinforcement Learning and the Q–learning algorithm

Consider an autonomous agent interacting with its environment via perception and action. On each interaction step the agent senses the current state s of the environment, and chooses an action a to perform. The action a alters the state s of the environment, and a scalar reinforcement signal r (a reward or penalty) is provided to the agent to indicate the desirability of the resulting state.

The goal of the agent in a RL problem is to learn an action policy that maximizes the expected long term sum of values of the reinforcement signal, from any starting state. A policy $\pi : \mathcal{S} \to \mathcal{A}$ is some function that tells the agent which actions should be chosen, under which circumstances [5]. This problem can be formulated as a discrete time, finite state, finite action Markov Decision Process (MDP). The learner's environment can be modeled [6] 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.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$: is a state transition function, where $\Pi(\mathcal{S})$ is a probability distribution over $\mathcal{S}$. $\mathcal{T}(s, a, s')$ represents the probability of moving from state s to s' by performing action a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \Re$: is a scalar reward function.

The task of a RL agent is to learn an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maps the current state s into an optimal action(s) a to be performed in s. In RL, the policy π should be learned through trial-and-error interactions of the agent with its environment, that is, the RL learner must explicitly explore its environment.

The Q–learning algorithm was proposed by Watkins [10] as a strategy to learn an optimal policy π^* when the model ($\mathcal{T}$ and $\mathcal{R}$) is not known in advance.

Let $Q^*(s,a)$ be the reward received upon performing action a in state s, plus the discounted value of following the optimal policy thereafter:

$$Q^*(s,a) \equiv R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s'). \tag{1}$$

The optimal policy π^* is $\pi^* \equiv \arg\max_a Q^*(s,a)$. Rewriting $Q^*(s,a)$ in a recursive form:

$$Q^*(s,a) \equiv R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a'} Q^*(s',a'). \tag{2}$$

Let $\hat{Q}$ be the learner's estimate of $Q^*(s,a)$. The Q–learning algorithm iteratively approximates $\hat{Q}$, i.e., the $\hat{Q}$ values will converge with probability 1 to Q^*, provided the system can be modeled as a MDP, the reward function is bounded $(\exists c \in \mathcal{R}; (\forall s,a), |R(s,a)| < c)$, and actions are chosen so that every state-action pair is visited an infinite number of times. The Q learning update rule is:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right], \tag{3}$$

where s is the current state; a is the action performed in s; r is the reward received; s' is the new state; γ is the discount factor $(0 \leq \gamma < 1)$; α is the learning rate.

An interesting property of Q–learning is that, although the exploration-exploitation tradeoff must be addressed, the $\hat{Q}$ values will converge to Q^*, independently of the exploration strategy employed (provided all state-action pairs are visited often enough) [6].

3 The Heuristically Accelerated Q–Learning Algorithm

The Heuristically Accelerated Q–Learning algorithm [1] was proposed as a way of solving the RL problem which makes explicit use of a heuristic function $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ to influence the choice of actions during the learning process. $H_t(s_t, a_t)$ defines the heuristic, which indicates the importance of performing the action a_t when in state s_t.

The heuristic function is strongly associated with the policy: every heuristic indicates that an action must be taken regardless of others. This way, it can be said that the heuristic function defines a "Heuristic Policy", that is, a tentative policy used to accelerate the learning process. It appears in the context of this paper as a way to use the knowledge about the policy of an agent to accelerate the learning process. This knowledge can be derived directly from the domain (prior knowledge) or from existing clues in the learning process itself.

The heuristic function is used only in the action choice rule, which defines which action a_t must be executed when the agent is in state s_t. The action choice rule used in the HAQL is a modification of the standard $\epsilon - Greedy$ rule used in Q–learning, but with the heuristic function included:

$$\pi(s_t) = \begin{cases} \arg\max_{a_t} \left[\hat{Q}(s_t, a_t) + \xi H_t(s_t, a_t)\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise,} \end{cases} \tag{4}$$

where:

- $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$: is the heuristic function, which influences the action choice. The subscript t indicates that it can be non-stationary.
- ξ: is a real variable used to weight 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: the greater the value of p, the smaller is the probability of a random choice.
- a_{random} is a random action selected among the possible actions in state s_t.

As a general rule, the value of the heuristic $H_t(s_t, a_t)$ used in the HAQL must be higher than the variation among the $\hat{Q}(s_t, a_t)$ for a similar $s_t \in S$, so it can influence the choice of actions, and it must be as low as possible in order to minimize the error. It can be defined as:

$$H(s_t, a_t) = \begin{cases} \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + \eta & \text{if } a_t = \pi^H(s_t), \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

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

As the heuristic is used only in the choice of the action to be taken, the proposed algorithm is different from the original Q–learning only in the way exploration is carried out. The RL algorithm operation is not modified (i.e., updates of the function Q are as in Q–learning), this proposal allows that many of the conclusions obtained for Q–learning to remain valid for HAQL [1].

The use of a heuristic function made by HAQL explores an important characteristic of some RL algorithms: the free choice of training actions. The consequence of this is that a suitable heuristic speeds up the learning process, and if the heuristic is not suitable, the result is a delay which does not stop the system from converging to a optimal value.

4 Experiment in the RoboCup 2D Simulation domain

One experiment was carried out using the RoboCup 2D Soccer Server [7]: the implementation of a defense team, with a goalkeeper and a first defender (fullback) that have to learn how to minimize the number of goals scored by the opponent. In this experiment, the implemented team have to learn while playing against a team composed of two striker agents from the UvA Trilearn 2001 Team [2].

The space state of the learning agents is composed by its position in a discrete grid with N x M positions the agent can occupy, the position of the ball in the same grid and the direction the agent is facing. This grid is different for the

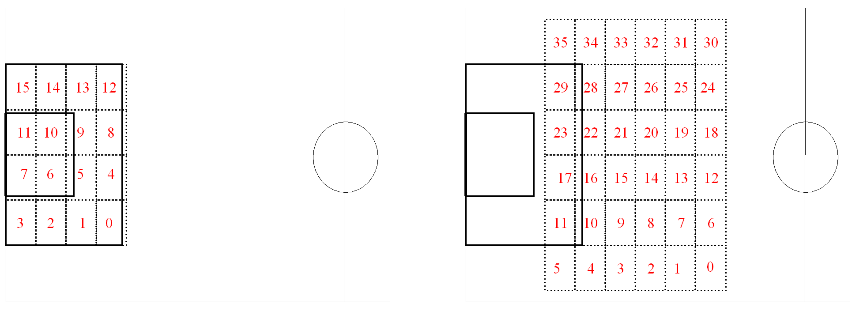

Fig. 1. Discrete grids that compose the space state of the goalkeeper (left) and the defender (right).

goalkeeper and the defender: each agent has a different area where it can move, which they cannot leave. These grids, shown in figure 1, are partially overlapping, allowing both agents to work together in some situations. The direction that the agent can be facing is also discrete, and reduced to four: north, south, east or west.

The defender can execute six actions: turnBodyToObject, that keeps the agent at the same position, but always facing the ball; interceptBall, that moves the agent in the direction of the ball; driveBallFoward, that allows the agent to move with the ball; directPass, that execute a pass to the goalkeeper; kickBall, that kick the ball away from the goal and; markOpponent, that moves the defender close to one of the opponents.

The goalkeeper can also perform six actions: turnBodyToObject, interceptBall, driveBallForward, kickBall, which are the same actions that the defender can execute, and two specific actions: holdBall, that holds the ball and moveToDefensePosition, that moves the agent to a position between the ball and the goal.

All these actions are implemented using pre-defined C++ methods defined in the BasicPlayer class of the UvA Trilearn 2001 Team. “The BasicPlayer class contains all the necessary information for performing the agents individual skills such as intercepting the ball or kicking the ball to a desired position on the field” [2, p. 50].

The reinforcement given to the agents were inspired on the definitions of rewards presented in [3], and are different for the agents. For the goalkeeper, the rewards consists of: ball caught, kicked or driven by goalie = 25; ball with any opponent player = -25; goal scored by the opponent = -100. For the defender, the rewards are: ball kicked or passed to the goalie = 15; ball with any opponent player = -10; goal scored by the opponent = -15.

The heuristic policy used for the goalkeeper and the defender is described by two rules: if the agent is not near the ball, run in the direction of the ball, and; if the agent is close to the ball, do something with it. Note that the heuristic policy is very simple, leaving the task of learning what to do with the ball and how to deviate from the opponent to the learning process. The values associated with the heuristic function are defined using equation 5, with the value of η set to 200. This value is computed only once, at the beginning of the game. In all the following episodes, the value of the heuristic is maintained fixed, allowing the learning process to overcome bad indications.

In order to evaluate the performance of the HAQL algorithm, this experiment was performed with teams of agents that learns using the Q–learning algorithm, the HAQL algorithm and using agents that acts based only on a heuristic rule (without learning capabilities). The results presented are based on the average of 10 training sessions for each algorithm. Each session is composed of 100 episodes consisting of matches taking 3000 cycles each. During the simulation, when a teams scores a goal all agents are transferred back to a pre-defined start position.

The parameters used in the experiments were the same for the two algorithms, Q–learning and HAQL: the learning rate is $\alpha = 1.25$, the exploration/ exploitation rate $p = 0.05$ and the discount factor $\gamma = 0.9$. Values in the Q table were randomly initiated, with $0 \leq Q(s, a, o) \leq 1$. The experiments were programmed in C++ and executed in a Pentium IV 2.8GHz, with 1GB of RAM on a Linux platform.

Figure 2 shows the learning curves for both algorithms when the agents learn how to play against a team composed of two strikers from the UvA Trilearn Team 2001 [2]. It presents the average goals scored by the opponent team in each episode. It is possible to verify that Q–learning has worse performance than HAQL at the initial learning phase, and that as the matches proceed, the performance of both algorithms become more similar, as expected.

Another important information contained in this figure is the number of goals scored against a defense team that uses only the heuristic policy to select which action must be done, at a given time. As it can be seen, this team will receive an average of 24 goals in each episode, performing worst than any of the other two algorithms. This shows that the heuristic policy by itself is not a complete solution to the problem, but only an indication of some actions that should be taken, at certain times.

Student's t–test [8] was used to verify the hypothesis that the use of heuristics speeds up the learning process. For the experiments described in this section, the value of the module of T was computed for each episode using the same data presented in figure 2. The result, presented in figure 3, shows that HAQL performs clearly better than Q–learning until the 60th episode, with a level of confidence greater than 95%. After the 60th episode, the results became closer. But it can be seen that HAQL still performs better than Q–learning.

Finally, table 1 shows the cumulative number of goals made by the strikers at the end of 100 episodes (averaged for 10 training sessions). What stands out in this tables is that, due to a lower number of goals scored against the HAQL at the beginning of the learning process, this algorithm receives significanly less goals than the Q–learning algoritm (with a statistical confidence $> 99.9\%$).

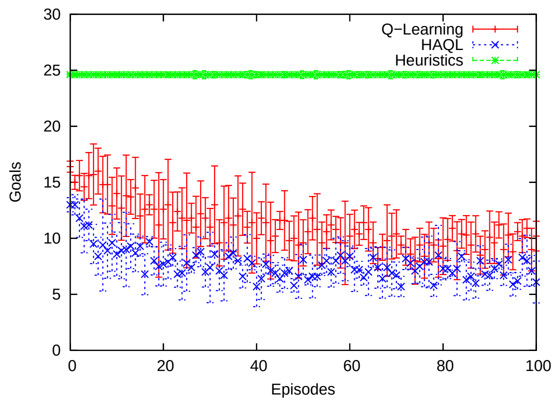

Fig. 2. Average goals scored against the defense agents using the Q–Learning and the HAQL algorithms, for training sessions against two UvA Trilearn attack agents.

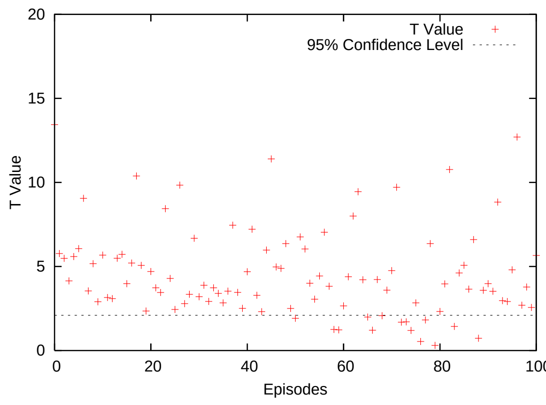

Fig. 3. Results from Student's t test between Q–learning and HAQL algorithms, for defense agents training against two UvA Trilearn attack agents.

5 Conclusion and Future Works

This paper presented the use of the Heuristically Accelerated Q–Learning (HAQL) algorithm to speed up the learning process of teams of mobile autonomous robotic agents acting in the RoboCup 2D Simulator.

The experimental results obtained in this domain showed that agents using the HAQL algorithm learned faster than ones using the Q–learning, when they were trained against the same opponent. These results are strong indications that the performance of the learning algorithm can be improved using very simple heuristic functions.

Due to the fact that the reinforcement learning requires a large amount of training episodes, the HAQL algorithm has been evaluated, so far, only in simu-

Table 1. Cumulative goal the end of 100 episodes (average for all training sessions).

Algorithm	Cumulative goal score
Q–leaning	(1177 ± 51)
HAQL	(836 ± 10)

lated domains. Among the actions that need to be taken for a better evaluation of this algorithm, the more important ones include:

- The development of teams composed of agents with more complex space state representation and with a larger number of players.
- Working on obtaining results in more complex domains, such as RoboCup 3D Simulation and Small Size League robots [4].
- Comparing the use of more convenient heuristics in these domains.
- Validate the HAQL by applying it to other the domains, such as the "car on the hill" and the "cart-pole".

References

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

[2] R. de Boer and J. Kok. *The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team.* Master's Thesis, University of Amsterdam, 2002.

[3] S. Kalyanakrishnan, Y. Liu, and P. Stone. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X.* Springer Verlag, Berlin, 2007.

[4] H. Kitano, A. Minoro, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: A challenge problem for ai. *AI Magazine*, 18(1):73–85, 1997.

[5] M. L. Littman and C. Szepesvári. A generalized reinforcement learning model: Convergence and applications. In *Procs. of the Thirteenth International Conf. on Machine Learning (ICML'96)*, pages 310–318, 1996.

[6] T. Mitchell. *Machine Learning.* McGraw Hill, New York, 1997.

[7] I. Noda. Soccer server : a simulator of robocup. In *Proceedings of AI symposium of the Japanese Society for Artificial Intelligence*, pages 29–34, 1995.

[8] M. R. Spiegel. *Statistics.* McGraw-Hill, 1998.

[9] C. Szepesvári and M. L. Littman. Generalized markov decision processes: Dynamic-programming and reinforcement-learning algorithms. Technical report, Brown University, Department of Computer Science, Brown University, Providence, Rhode Island 02912, 1996. CS-96-11.

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