Heuristically Accelerated Q–Learning: a new approach to speed up Reinforcement Learning

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

[1] Laboratório de Técnicas Inteligentes
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav. 3, 158. 05508-900 São Paulo - SP, Brazil.
[2] Instituto Tecnológico de Aeronáutica
Praça Mal. Eduardo Gomes, 50 –
12228-900 – São José dos Campos – SP
reinaldo.bianchi@poli.usp.br, carlos@ita.br, anna.reali@poli.usp.br

Abstract. This work presents a new algorithm, called Heuristically Accelerated Q–Learning (HAQL), that allows the use of heuristics to speed up the well-known Reinforcement Learning algorithm Q–learning. A heuristic function $\mathcal{H}$ that influences the choice of the actions characterizes the HAQL algorithm. The heuristic function is strongly associated with the policy: it indicates that an action must be taken instead of another. This work also proposes an automatic method for the extraction of the heuristic function $\mathcal{H}$ from the learning process, called Heuristic from Exploration. Finally, experimental results shows that even a very simple heuristic results in a significant enhancement of performance of the reinforcement learning algorithm.
Keywords: Reinforcement Learning, Cognitive Robotics.

1 Introduction

The main problem approached in this paper is the speedup of Reinforcement Learning (RL), aiming its use in mobile and autonomous robotic agents acting in complex environments. The goal of this work is to propose an algorithm that preserves RL advantages, such as the convergence to an optimal policy and the free choice of actions to be taken, minimizing its main disadvantage: the learning time.

For being the most popular RL algorithm and because of the large amount of data available in literature for a comparative evaluation, the Q–learning algorithm [11] was chosen as the first algorithm to be extended by the use of heuristic acceleration. The resulting new algorithm is named Heuristically Accelerated Q–Learning (HAQL) algorithm.

In order to describe this proposal in depth, this paper is organized as follows. Section 2 describes the Q–learning algorithm. Section 3 describes the HAQL and its formalization using a heuristic $\mathcal{H}$ function and section 4 describes the algorithm used to define the heuristic function, namely Heuristic from Exploration. Section 5 describes the domain where this proposal has been evaluated and the

results obtained. Finally, Section 6 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 $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is some function that tells the agent which actions should be chosen, under which circumstances [8]. This problem can be formulated as a discrete time, finite state, finite action Markov Decision Process (MDP), since problems with delayed reinforcement are well modeled as MDPs. The learner's environment can be modeled (see [7, 9]) 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.

The task of a RL agent is to learn an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maps the current state s into a desirable 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 [11] 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 ($0 \leq \gamma < 1$); $\alpha = 1/(1 + visits(s, a))$, where $visits(s, a)$ is the total number of times this state-action pair has been visited up to and including the current iteration.

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) [9].

3 The Heuristically Accelerated Q–Learning Algorithm

The Heuristically Accelerated Q–Learning algorithm can be defined 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 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:

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

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

For instance, if the agent can execute 4 different actions when in state s_t, the values of $\hat{Q}(s_t, a)$ for the actions are [1.0 1.1 1.2 0.9], the action that the heuristic suggests is the first one. If $\eta = 0.01$, the values to be used are $H(s_t, 1) = 0.21$, and zero for the other actions.

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.

Theorem 1. *Consider a HAQL agent learning in a deterministic MDP, with finite sets of states and actions, bounded rewards* $(\exists c \in \mathcal{R}; (\forall s, a), |R(s, a)| < c)$*, discount factor* γ *such that* $0 \leq \gamma < 1$ *and where the values used on the heuristic function are bounded by* $(\forall s_t, a_t)$ $h_{min} \leq H(s_t, a_t) \leq h_{max}$*. For this agent, the* $\hat{Q}$ *values will converge to* Q^**, with probability one uniformly over all the states* $s \in \mathcal{S}$*, if each state-action pair is visited infinitely often (obeys the Q–learning infinite visitation condition).*

Proof: In HAQL, the update of the value function approximation does not depend explicitly on the value of the heuristic. The necessary conditions for the convergence of Q–learning that could be affected with the use of the heuristic algorithm HAQL are the ones that depend on the choice of the action. Of the conditions presented in [8, 9], the only one that depends on the action choice is the necessity of infinite visitation to each pair state-action. As equation 4 considers an exploration strategy ϵ– *greedy* regardless of the fact that the value function is influenced by the heuristic – $\hat{Q} + H$ –, the infinite visitation condition is guaranteed and the algorithm converges. *q.e.d.*

The condition of infinite visitation of each state-action pair can be considered valid in practice – in the same way that it is for Q–learning – also by using other visitation strategies:

- Using a Boltzmann exploration strategy [7].
- Intercalating steps where the algorithm makes alternate use of the heuristic and exploration steps.
- Using the heuristic during a period of time, smaller than the total learning time for Q–learning.

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.

The idea of using heuristics with a learning algorithm has already been considered by other authors, as in the Ant Colony Optimization presented in [5, 2]. However, the possibilities of this use were not properly explored yet. The complete HAQL algorithm is presented on table 1. It can be noticed that the only difference to the Q–learning algorithm is the action choice rule and the existence of a step for updating the function $H_t(s_t, a_t)$.

Table 1. The HAQL algorithm.

Initialize $Q(s, a)$.
Repeat:
Visit the s state.
Select an action a using the action choice rule (equation 4).
Receive the reinforcement $r(s, a)$ and observe the next state s'.
Update the values of $H_t(s, a)$.
Update the values of $Q(s, a)$ according to:
$Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$.
Update the $s \leftarrow s'$ state.
Until some stop criteria is reached.

where: $s = s_t$, $s' = s_{t+1}$, $a = a_t$ e $a' = a_{t+1}$.

Although any function which works over real numbers and produces values belonging to an ordered set may be used in equation 4, the use of addition is particularly interesting because it allows an analysis of the influence of the values of $\mathcal{H}$ in a way similar to the one which is made in informed search algorithm (such as A^* [6]).

Finally, the function $H_t(s_t, a_t)$ can be derived by any method, but a good one increases the speedup and generality of this algorithm. In the next section, the method Heuristic from Exploration is presented.

4 The method Heuristic from Exploration

One of the main questions addressed in this paper is how to find out, in an initial learning stage, the policy which must be used for learning speed up. For the HAQL algorithm, this question means how to define the heuristic function. The definition of an initial situation depends on the domain of the system application. For instance, in the domain of robotic navigation, we can extract an

useful heuristic from the moment when the robot is receiving environment reinforcements: after hitting a wall, use as heuristic the policy which leads the robot away from it.

Along with HAQL a method named Heuristic from Exploration was used. This method is composed of two phases: the first one, which extracts information about the structure of the environment through exploration and the second one, which defines the heuristic for the policy, using the information extracted. These stages were called Structure Extraction and Heuristic Backpropagation, respectively.

Structure Extraction iteratively estimates the state transition function $\hat{t}_t(s_t, a_t, s_{t+1})$, keeping track of the result from all the actions executed by the agent.

In the case of a mobile robot, when the agent tries to move from one position to the next, the result of the action is recorded. When a action does not result in a move, it indicates the existence of an obstacle in the environment. With the passing of the time this method generates a map sketch of the environment and possible actions in each state.

From the model of the environment, the Heuristic Backpropagation composes the heuristic. It propagates – from a final state – the correct policies which lead to that state. For instance, the heuristic of the states immediately previous to a terminal state are defined by the actions that lead to the terminal state. In a following iteration, this heuristic is propagated to the predecessors of the states which already have a defined heuristic and so on.

Theorem 2. *For a deterministic MDP whose model is known, the Heuristic Backpropagation algorithm generates an optimal policy.*

Proof: As this algorithm is a simple application of the Dynamic Programming algorithm [1], Bellman's theorem proves this statement. *q.e.d.*

The Heuristic Backpropagation is an algorithm very similar to the Dynamic Programming algorithm [1]. In case where the environment is completely known, both of them work the same way. In case where only part of the environment is known, the backpropagation is done only for the known states. On the example of robotic mapping, where the model of the environment is gradually built, the backpropagation can be done only on the parts of the environment which are already mapped.

Results for a complete implementation of this algorithm will be presented in the next section.

5 Experiments in the mobile robots domain

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 a simulated domain.

In these experiments, a mobile robot that can move in four directions have to find a specific state, the target. The environment is discretized in a grid with N x

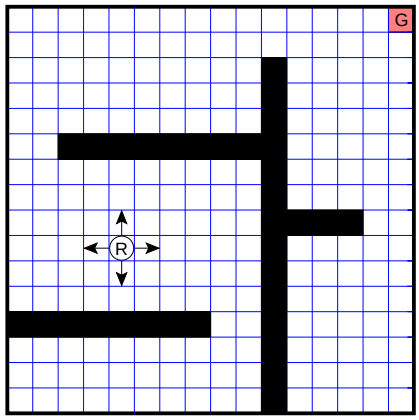

Fig. 1. Room with walls (represented by dark lines) discretized in a grid of states.

M positions the robot can occupy. The environment in which the robot moves can have walls (figure 1), represented by states to which the robot cannot move. The robot can execute four actions: move north, south, east or west. This domain, called *grid-world*, is well-known and was studied by several researchers [7, 9, 3, 4]. Two experiments were done using HAQL with Heuristic from Exploration in this domain: navigation with goal relocation and navigation in a new and unknown environment.

The value of the heuristic used in HAQL is defined using equation 5 as:

$$H(s_t, a_t) = \begin{cases} \max_a \hat{Q}(s_t, the) - \hat{Q}(s_t, a_t) + 1 & \text{se } a_t = \pi^H(s_t), \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

This value is computed only once, in the beginning of the acceleration. In all the following episodes, the value of the heuristic is maintained fixed, allowing the learning to overcome bad indications. If $H(s_t, a_t)$ is recalculated at each episode, a bad heuristic would be difficult to overcome.

For comparative effects, the same experiments are also executed using the Q–learning. The parameters used in Q–learning and HAQL were the same: learning rate $\alpha = 0.1$, the exploitation/exploration rate is 0.9 and the discount factor $\gamma = 0.99$. The rewards used were +10 when the robot arrives to the goal state and -1 when it executes any action. All the experiments presented were encoded in C++ Language and executed in a Pentium 3-500MHz, with 256MB of RAM, and Linux operating system.

The results presented in the next sub-sections show the average of 30 training sessions in nine different configurations of the navigation environment – a room with several walls – similar to the one in figure 1. The size of the environment is of 55×55 positions and the goal is initially at the right superior corner. The robot always start at a random position.

5.1 Robot navigation with goal relocation

In this experiment the robot must learn to reach the goal, which is initially located at the right superior corner (figure 1) and, after a certain time, is moved to the left inferior corner of the grid.

The HAQL initially only extracts the structure of the problem (using the structure extraction method described in section 4), behaving as the Q–learning. At the end of 4999^{th} episode the goal is relocated. With this, both algorithms must find the new position of the goal. As the algorithms are following the politics learned until then, the performance worsens and both algorithms execute a large number of steps to reach the new position of the goal.

As the robot controlled by the HAQL arrives at the new goal position (at the end of 5000^{th} episode), the heuristic to be used is constructed using the Heuristic Backpropagation (described in section 4) with information from the structure of the environment (that was not modified) and the new position of the goal, and the values of $H(s_t, a_t)$ are defined. This heuristic then is used, resulting in a better performance in relation to Q–learning, as shown in figure 2.

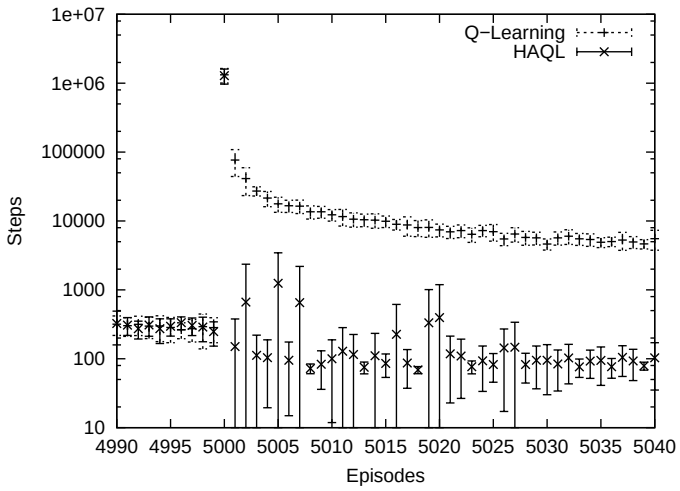

Fig. 2. Result for the goal relocation at the end of the 4999^{th} episode (log y).

It can be observed that the HAQL has a similar performance to Q–learning until the 5000^{th} episode. In this episode, the robot controlled by both algorithms takes more than 1 million steps to find the new position of the goal (since the known politics takes the robot to a wrong position).

After the 5001^{th} episode, while the Q–learning needs to learn the politics from scratch, the HAQL will always execute the minimum number of steps necessary to arrive at the goal. This happens because the heuristic function allows the HAQL to use the information about the environment it already possessed.

5.2 Robot navigation in a new environment

In the second experiment the robot must learn to reach the goal located at the right superior corner (figure 1) when inserted in an unknown environment, at a random position.

Again, the HAQL initially only extracts the structure of the problem, without making use of the heuristic, and behaving as the Q–learning. At the end of the ninth episode, the heuristic to be used is constructed using the Heuristic Backpropagation with the information from the structure of the environment extracted during the first nine episodes, and the values of $H(s_t, a_t)$ are defined. This heuristic is then used in all the following episodes.

The result (figure 3) shows that, while the Q–learning continue to learn the action policy, the HAQL converges to the optimal policy after the speed up.

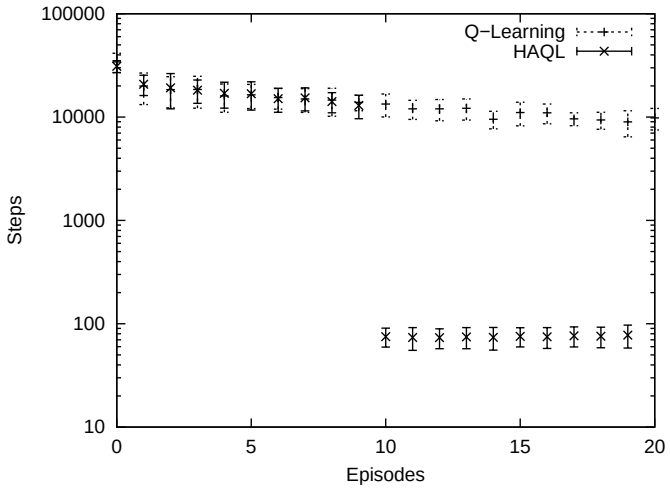

Fig. 3. Result for the acceleration at the end of the 9^{th} episode (log y).

The 10^{th} episode was chosen for the beginning of the acceleration because this allows to the agent to explore the environment before using the heuristic. As the robot starts every episode at a random position and the environment is small, the Heuristic from Exploration method will probably define a good heuristic.

Finally, Student's t–test [10] was used to verify the hypothesis that the use of heuristics speed up the learning process. For both experiments described in this section – goal relocation and navigation in a new environment – the value of the module of T was calculated for each episode using the same data presented in figures 2 and 3. The results confirm that after the speed up the algorithms are significantly different, with a confidence level greater than 0.01%.

6 Conclusion and Future Works

This work presented a new algorithm, called Heuristically Accelerated Q–Learning (HAQL), that allows the use of heuristics to speed up the well-known Reinforcement Learning algorithm Q–learning.

The experimental results obtained using the automatic method for the extraction of the heuristic function $\mathcal{H}$ from the learning process, called Heuristic from Exploration, showed that the HAQL attained better results than the Q–learning for the domain of mobile robots.

Among the actions that need to be taken for a better evaluation of this proposal, the more important ones are:

- Validate the HAQL, by applying it to other the domains such as the "car on the hill" [3] and the "cart-pole" [4].
- During this study several indications that there must be a large number of methods which can be used to extract the heuristic function were found. Therefore, the study of other methods for heuristic composition is needed.

References

[1] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Upper Saddle River, NJ, 1987.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behaviour. *Nature 406 [6791]*, 2000.

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

[4] D. Foster and P. Dayan. Structure in the space of value functions. *Machine Learning*, 49(2/3):325–346, 2002.

[5] L. Gambardella and M. Dorigo. Ant–Q: A reinforcement learning approach to the traveling salesman problem. *Proceedings of the ML-95 – Twelfth International Conference on Machine Learning*, pages 252–260, 1995.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[8] 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.

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

[10] U. Nehmzow. *Mobile Robotics: A Practical Introduction*. Springer-Verlag, Berlin, Heidelberg, 2000.

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