The use of heuristics to speedup Reinforcement Learning

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

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

[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 class of algorithms that allows the use of heuristics based on policy to speed up reinforcement learning algorithms. This class of algorithms, called "Heuristically Accelerated Learning" is modeled by a convenient mathematical formalism known as Markov Decision Processes. An heuristic function $\mathcal{H}$ that influences the choice of the actions is defined in order to model the HALs. This heuristic is updated during the learning process. This work also proposes automatic methods for the extraction of the heuristic function $\mathcal{H}$ from the problem domain or from the learning process, called "Heuristic from X". A new algorithm called Heuristically Accelerated Q–Learning is proposed to validate this work. It implements a HAL by extending the well-known RL algorithm Q–Learning. Finally, we show some experimental results that allow to conclude that even a very simple heuristic results in a significant increase of the performance of the reinforcement learning algorithm used. A list of future work is then presented.*

1. Introduction

The main problem approached in this paper is the speedup of the Reinforcement Learning (RL) aiming its use in mobile and autonomous robotic agents acting in complex environments. We wish to keep the RL algorithms advantages, including the convergence for the stationery policy, the free choice of actions to be taken, and the unsupervised learning, minimizing its main disadvantage: the time necessary for learning.

The main hypothesis of this paper is that there is a class of RL algorithms that allows the use of heuristics to approach the problem of learning speedup. This class of algorithms is called *Heuristically Accelerated Learning* – (HAL).

The rest of this paper describes this proposal deeply: section 2 describes the meta-algorithm of "Heuristically Accelerated Learning"; 3 describes the formalization of HALs using a heuristic $\mathcal{H}$ function and section 4 describes the algorithms used to define the heuristic function, named "Heuristic from X"; sections 5 and 6 describe a complete implementation of an algorithm of HAL class based on Q-Learning, the Heuristically Accelerated Q-Learning – HAQL; finally 7 describes the domain where this proposal is being evaluated and the results obtained.

2. The HAL Meta-Algorithm

In a more formal way, a HAL class algorithm can be defined as a way of solving a modeling problem by a MDP which definitely sees the heuristic $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ to influence

the actions choice, by the agent during its learning. $H_t(s_t, a_t)$ defines the heuristic which indicates the importance of performing the action a_t in s_t state.

The heuristic function is strongly associated to the policy: every heuristic indicates that an action must be taken despite other. This way, we can say that the heuristic function defines a "Heuristic Policy", that is, an attempt policy used to accelerate the learning process.

As the heuristic is used only for the choice of the action to be taken, this algorithms class is different from the RL algorithms proposed so far just for the way the explanation is carried out. As the RL algorithm operation isn't modified (for instance, without changing the updating of the function Q), this proposal allows many of the obtained conclusions of RL algorithms to remain valid for the HALs. [1] The use of heuristic function made by HALs explores an important characteristic of some RL algorithms: the free choice of training actions. The consequence of this user is that a suitable heuristics speeds up the learning process, and if the heuristic isn't suitable, the result is a delay which doesn't stop the system from converging to a optimal value (in case of the deterministic systems).

Another important HALs feature is that the heuristic function can be modified on each iteration. This allows the speedup to be used in a premature stage of Learning and modified each time more information about the system becomes available.

The class of the algorithms "Heuristically Accelerated Learning"can be divided in two sub-classes: the one which uses the knowledge about the domain to abstract a heuristic and the ones which use clues that exist in the learning process itself.

In the first case, the heuristic is learned from the states noticed by the agent. This learning process can be made by generic techniques or "ad hoc"methods. Like this, the heuristic is a way of knowledge generalization which we have concerning the domain. [2].

In the second case, the heuristic is extracted from data of the learning process itself. Among these clues, two of them are more relevant: the value function and the system policy in a certain moment.

A second hypothesis made in this paper is that these must be a large amount of methods which can be used to learn the heuristic function. Since these are many domains in which RL can be used and many ways of extracting knowledge from a domain, this hypothesis is easily validated. The learning methods of the heuristic policy are named *"Heuristic from X""*.

The generic procedure which defines the operation of the "Heuristically Accelerated Learning"can be described as a meta-algorithm. This procedure involves four steps repeated sequentially until some stop criteria is achieved. This meta-algorithm is described in table 1. [3]

Any algorithm that uses heuristics to select an action, which can be estimated *on-line* is an instance of the meta-algorithm HAL. This way, we can build algorithms of this class from most reinforcement learning algorithms. As an example in section 5 the *Heuristically Accelerated Q–Learning*, will be presented, extending the Q-Learning

[1] We can say that the reinforcement learning algorithms are a subgroup of HAL algorithms, where the influence of heuristic is always null.

[2] As in the optimization by ant colonies, [Bonabeau et al., 2000] which uses the distance of the cities as heuristic.

[3] This nomenclature is made because as there are many techniques to estimate the shape of the objects in the area of Computational Vision (CV), called *"Shape from X"*, here there are also many ways to estimate the heuristic. In the "Shape from X" the individual techniques are named as "Shape from Shading", "Shape from Texture", "Shape from Stereo' [Nalwa, 1993]. .

Tabela 1: The meta-algorithm "Heuristically Accelerated Learning"

Initialize the estimate of the value function.
Repeat:
Being under s_t state select an action using a
combination of the heuristic with a suitable value function.
Receive the reinforcement $r(s, a)$ and notice the next state s'.
Update the heuristic function $H_t(s, a)$ using a suitable
"method Heuristic from X"
Update the values of the value function used.
Update the state $s \leftarrow s'$.
Until some stop criteria is achieved.

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

algorithm [Watkins, 1989].

The idea of using heuristics with a learning algorithm has already been approached by other authors, as in the approaching of Optimization by Ant Colony presented in [Gambardella and Dorigo, 1995, Bonabeau et al., 2000]. However, the possibilities of this use weren't properly explored yet. Particularly, the use of heuristics extracted following a similar methodology to the one proposed by [Drummond, 2002] seems very promising. Afterwards, each element of the HAL meta-algorithm is analyzed deeply.

3. The Heuristic function $\mathcal{H}$

The heuristic function appears in the context of this paper as the 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 or built from existing clues in the learning process itself.

The heuristic function is used only in the transition of states which chooses the action a_t to be taken in the state s_t. This way, we can use, to this class of algorithms, the same formalism used on the RL. A strategy to the action choice is the random exploration $\epsilon - Greedy$, in which not only the estimate of the probability of transition functions $\mathcal{T}$, and the reward $\mathcal{R}$, but also the function $\mathcal{H}$ is bore in mind. The rule of this transition states is given by:

$$\pi(s_t) = \begin{cases} \arg\max_{a_t} \left[\mathsf{F}(s_t, a_t) \bowtie \xi H_t(s_t, a_t)^\beta\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise}, \end{cases} \quad (1)$$

where:

- $\mathcal{F} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$: is an estimate of a value function which describes the accumulated expected reward. For instance, if $\mathsf{F}(s_t, a_t) \equiv \hat{Q}(s_t, a_t)$ we have an algorithm similar to the *Q–Learning*. Another option is to use $\mathsf{F}(s_t, a_t) \equiv r(s_t, a_t) + \gamma \hat{V}(s_{t+1})$, resulting in an extension of the algorithm of Temporal Differences[Sutton, 1988].
- $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$: is the heuristic function, which influences the choice of action, defining the importance of executing the action a_t at being on state s_t. The note t in heuristic function indicates that it can be different in two different moments.
- $\bowtie$: is a mathematical function, which must operate on numbers and produce a value belonging to a ordered set (which the operation of maximization can be applied).

- ξ and β: are real variables used to weight the influence of the heuristic function.
- q is a value chosen in an random way with uniform probability in [0,1] and p ($0 \leq p \leq 1$) is the parameter which defines the exploration/exploitation ratio: the bigger the value of p, the smaller is the probability of an random choice.
- a_{random} is an random action selected among the executable actions in state s_t.

The first consequence of this formulation is that if ξ or β reduce with time more quickly than the learning rate, the convergence evidences existing for the RL algorithms remain valid in this approach. Using *Q–Learning* as an example ($\bowtie$ = addition), if the decrease of ξ is faster than that of the learning rate α, the algorithm converges. Afterwords two theorems will be presented to prove this statement.

Theorem 1 *The use of heuristics in a HAL acting in a deterministic MDP, with groups of finite states and actions, limited rewards* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, discount factor* γ *the way* $0 \leq \gamma < 1$ *and used values on the heuristic function limited* $(\forall s_t, a_t)\ h_{min} \leq H(s_t, a_t) \leq h_{max}$*, doesn't produce infinite values on the approach of value function.*

Proof: In HAL algorithms, the updating of the approach doesn't depend on the value of the heuristic clearly. Thus, the influence that the heuristic has in the $\mathsf{F}(s_t, a_t)$ is caused by the differences which may occur during the exploration/exploitation.

Depending on the heuristic correctness, the exploration may have two effects: if the heuristic $H_t(s_t, a_t)$ estimates the grid policy perfectly, the exploration is reduced, because the agent always tends to follow less reinforcements, but there's no effect which produces infinite values in $\mathsf{F}(s_t, a_t)$.

If the heuristic $H_t(s_t, a_t)$ is incorrect, the agent will explore states which it wouldn't without its influence. The only possibility where an infinite value would be generated in the approach of the value function is the one which the agent is blocked in a similar state (in *deadlock*), always executing the same action.

However, in this case, the action executed by the agent doesn't change the estimate of the value function. [4] That's because, if the action changed F, after a finite number of actions the heuristic, which limited by $h_{min} \leq H(s_t, a_t) \leq h_{max}$, would not be considered. The approach of the value function would be so unfavorable that, even if with the influence of the heuristic, this action wouldn't be chosen anymore leaving the *deadlock* [5]. □

Theorem 2 *If the technique of Reinforcement Learning in which a HAL is based converges independently of the initialization, the HAL also converges.*

Proof: In most convergence tests of the algorithms of reinforcement learning, the only restriction for the initial value of the function value approach is that they must be finite(see [Mitchell, 1997, pg. 378] for the Q-Learning, [Szepesvári and Littman, 1996] for other models).

As the use of the heuristic doesn't produce infinite values on the function value approach (theorem 1), there must be only a δt delay time between the end of the heuristic influence (decay of ξ or β up to zero) and the decrease of the learning rate α, where this δt is enough time for the learning, to occur, which the HAL converges. □

[4] This can occur because the value of F is already on the lower possible bound for that action or because the action doesn't generate rewards.

[5] This theorem doesn't prove that the system never is in *deadlock*, but even if this happens, the values of $\mathsf{F}(s_t, a_t)$ won't tend to infinite

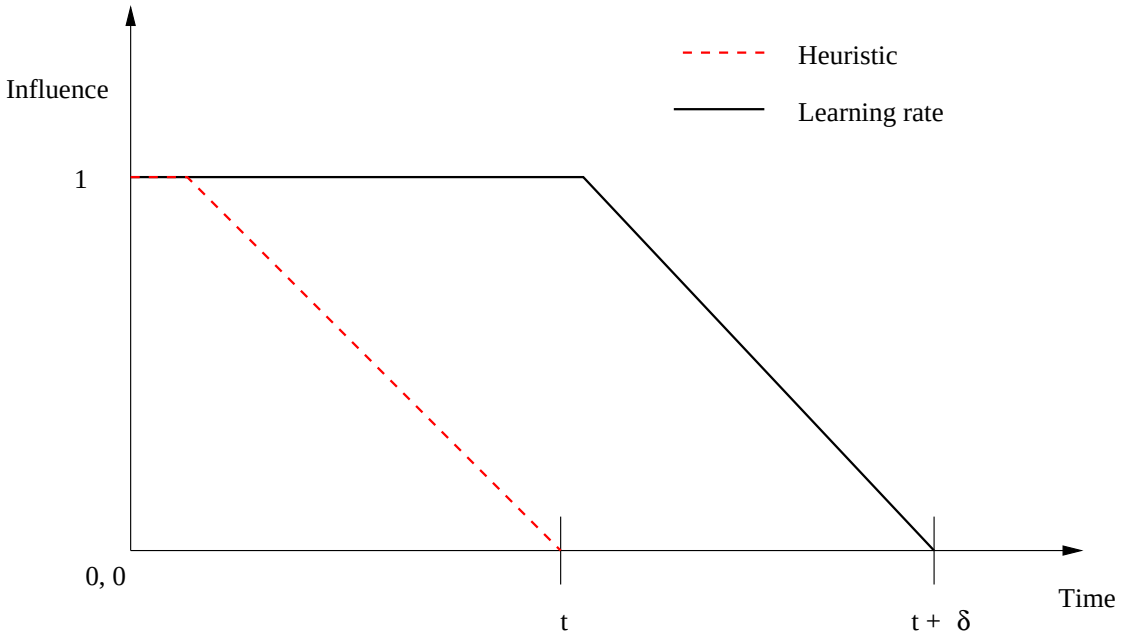

Figura 1: If δ is enough for learning, the system converges.

In the extreme case, where the learning rate never decays ($\delta = \infty$), this result is trivial. In the case where both the heuristic influence and the learning rate decay, the heuristic influence must end before the learning itself. Figure 1 shows an example where this happens.

These two theorems restrict the value which can be used on the heuristic for finite numbers. The next theorem allows limiting the values to be used in the heuristic function even more accurately.

Definition 1 *The error caused in the approach of the value function due to the use of the heuristic on learning algorithm is defined by*

$$L_H(s_t) = \mathsf{F}(s_t, \pi^*) - \mathsf{F}(s_t, \pi^H), \forall s_t \in S, \tag{2}$$

where $\mathsf{F}(s_t, \pi^H)$ is the estimate of the value function calculated from the suitable policy by heuristic, π^H.

Corollary 1 *In a system in valance on the optimal value, the use of correct heuristic won't cause any changes in the system.*

Proof: Direct consequence of the equation 2,because in the cause when the heuristic makes the used policy coincide with the optimal policy, we have $\pi^H = \pi^*$, $\mathsf{F}(s_t, \pi_H) = \mathsf{F}(s_t, \pi^*)$ and therefore the error $L_H(s_t) = 0$. □ The theorem presented afterwards defines the superior bound for the error $L_H(s_t), \forall s_t \in S$.

Theorem 3 *The maximum error caused by the use of a limited heuristic by $h_{min} \leq H(s_t, a_t) \leq h_{max}$ in a HAL acting in a deterministic MDP, with sets of finite states and actions, limited rewards $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$, discount factor γ the way $0 \leq \gamma < 1$ and which function $\bowtie$ is used is the sum is superior bounded by:*

$$L_H(s_t) \leq \xi\ [h^{\beta}_{max} - h^{\beta}_{min}]. \tag{3}$$

Proof: This proof will be done in parts, because $\mathsf{F}(x_t)$ can be defined in two different ways:

1st. Case: $\mathsf{F}(s_t, a_t) \equiv Q(s_t, a_t)$.

The value function $Q(s_t, a_t)$ can be defined as [Ribeiro, 2002]:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) V^*(s_{t+1}). \tag{4}$$

where:

- s_t is the present state,
- a_t is the action performed in s_t,
- s_{t+1} is the state resulting of applying the action a_t at being in s_t,
- $T(s_t, a_t, s_{t+1})$ is the function of transition probability and
- V is the value function.

In case of using a HAL based on $Q(s_t, a_t)$, the equation (1) becomes:

$$\pi(s_t) = \begin{cases} \arg\max_{a_t} \left[r(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) V^*(s_{t+1}) + \xi H_t(s_t, a_t)^\beta \right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise.} \end{cases} \tag{5}$$

There's a z state which causes the maximum error: $\exists z \in S, \forall s \in S, L_H(z) \geq L_H(s)$. For this z state, consider the optimal action $a = \pi^*(z)$ and the suitable action by the heuristic $b = \pi^H$. As the action choice is made by an ϵ-*Greedy* policy, b must seem at least as good as a:

$$\begin{aligned} & r(z,a) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(z, a, s_{t+1}) V^*(s_{t+1}) + \xi H_t(z,a)^\beta \leq \\ & r(z,b) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(z, b, s_{t+1}) V^*(s_{t+1}) + \xi H_t(z,b)^\beta \end{aligned}$$

$$\begin{aligned} r(z,a) - r(z,b) \leq & \gamma \sum_{s_{t+1} \in \mathcal{S}} T(z, b, s_{t+1}) V^*(s_{t+1}) + \xi H_t(z,b)^\beta \\ & - \left[\gamma \sum_{s_{t+1} \in \mathcal{S}} T(z, a, s_{t+1}) V^*(s_{t+1}) + \xi H_t(z,a)^\beta \right] \end{aligned}$$

$$\begin{aligned} r(z,a) - r(z,b) \leq & \xi \left[H_t(z,b)^\beta - H_t(z,a)^\beta \right] \\ & + \gamma \sum_{s_{t+1} \in \mathcal{S}} \left[T(z, b, s_{t+1}) V^*(s_{t+1}) - T(z, a, s_{t+1}) V^*(s_{t+1}) \right]. \end{aligned} \tag{6}$$

The maximum error is:

$$\begin{aligned} L_H(z) &= \mathsf{F}(z, \pi^*) - \mathsf{F}(z, \pi^H) = Q(z,a) - Q(z,b) \\ L_H(z) &= r(z,a) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(z, a, s_{t+1}) V^*(s_{t+1}) \\ & \quad - \left[r(z,b) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(z, b, s_{t+1}) V^*(s_{t+1}) \right] \end{aligned}$$

$$\begin{aligned} L_H(z) = & r(z,a) - r(z,b) \\ & + \gamma \sum_{s_{t+1} \in \mathcal{S}} \left[T(z, a, s_{t+1}) V^*(s_{t+1}) - T(z, b, s_{t+1}) V^*(s_{t+1}) \right] \end{aligned} \tag{7}$$

Substituting the equation (6) in (7), we have:

$$\begin{aligned} L_H(z) &\leq \xi \left[H_t(z,b)^\beta - H_t(z,a)^\beta\right] \\ &\quad + \gamma \sum_{s_{t+1}\in\mathcal{S}} \left[T(z,b,s_{t+1})V^*(s_{t+1}) - T(z,a,s_{t+1})V^*(s_{t+1})\right] \\ &\quad + \gamma \sum_{s_{t+1}\in\mathcal{S}} \left[T(z,a,s_{t+1})V^*(s_{t+1}) - T(z,b,s_{t+1})V^*(s_{t+1})\right] \\ L_H(z) &\leq \xi \left[H_t(z,b)^\beta - H_t(z,a)^\beta\right]. \end{aligned} \tag{8}$$

Finally, as action b is chosen instead of action a, $H_t(z,b)^\beta \geq H_t(z,a)^\beta$. As the value of $\mathcal{H}$ is limited by $h_{min} \leq H(s_t,a_t) \leq h_{max}$, we can conclude:

$$L_H(s_t) \leq \xi\ [h_{max}^\beta - h_{min}^\beta], \forall s_t \in S. \quad \square \tag{9}$$

This proof is similar to the one presented in [Singh, 1994, p. 53], where it's demonstrated that small errors on the approaching of the of the function value can't produce arbitrarily bad results in a system based on policy iterations when the actions are selected in a *greedy* way.

2nd. Case: $\mathsf{F}(s_t,a_t) \equiv r(s_t,a_t) + \gamma\hat{V}(s_{t+1})$.

The reinforcement learning algorithms based on a policy iteration which directly use of the value function $V(s_t)$ to compute the policy, maximize the reinforcement sum $r(s,\pi(s))$ with the value $V^\pi(s')$ of the succeeding state (or its estimate), discounted of the γ:

$$\pi'(s_t) \leftarrow argmax_{\pi(s_t)} \left[r(s_t,\pi(s_t)) + \gamma \sum_{s_{t+1}\in\mathcal{S}} T(s_t,\pi(s_t),s_{t+1})V^\pi(s_{t+1})\right]. \tag{10}$$

A HAL based on this class of algorithms chooses the policy to be followed from the equation:

$$\pi'(s_t) \leftarrow argmax_{\pi(s_t)} \left[r(s_t,\pi(s_t)) + \gamma \sum_{s_{t+1}\in\mathcal{S}} T(s_t,\pi(s_t),s_{t+1})V^\pi(s_{t+1}) + \xi H(s_t,\pi(s_t))^\beta\right]. \tag{11}$$

We notice that the equation (11) is similar to the equation (5) and that every argument used to prove this theorem when $\mathsf{F}(s_t,a_t) \equiv Q(s_t,a_t)$ is also valid to this case. $\square$

Only a small algebraic manipulation is needed to prove that the error $L_H(s_t)$ has a defined upper limit in case the function $\bowtie$ is defined containing one of the four basic operations (trivial in case of subtraction). For other functions, this prove may not be possible.

As a general rule, the value of the heuristic $H_t(s_t,a_t)$ must be higher than the variation between $\mathsf{F}(s_t,a)$ for a similar $s_t \in S$, so it can influence the choice of actions and it must be as low as possible to minimize the error $L_H(s_t)$. In case the function $\bowtie$ used is the addition operation, the heuristic can be defined as:

$$H_t(s_t,a_t) = \begin{cases} \max_a\left[\mathsf{F}(s_t,a)\right] - \mathsf{F}(s_t,a_t) + \eta & \text{se } a_t = \pi^H(s_t), \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

where η is a low value and $\pi^H(s_t)$ is the heuristic for the obtained policy from a method "Heuristic from X".

For instance, if a state has 4 possible actions, the values of $\mathsf{F}(s_t, a)$ calculated for the actions are [1,0 1,1 1,2 0,9] and we wish the selected action is the first one, we can use $\eta = 0,01$, resulting in $H(s_t, 1) = 0,21$ and equal to zero for the other actions. The heuristic can be defined in a similar way for other function $\bowtie$.

The function $\bowtie$ is the last item introduced by the formulation presented in the equation 1. Although any function which works over real numbers (because both the function value such the heuristic have real values) and produces values belonging to an ordered set may be used, the most used ones are addition and multiplication. 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 (as the A^* [Russell and Norvig, 1995]), allowing a the reuse of many of the obtained results for this algorithm.

Finally, the multiplication may also be used instead of the function $\bowtie$. For instance, the rule of state transition of *Ant Colony System* ([Bonabeau et al., 2000]) uses the multiplication, where $H(s_t, a_t)$ is pondered by the β constant. However, a more detailed analysis shows that the use of multiplication may experience problems when the estimate function $\mathsf{F}(s_t, a_t)$ may admit positive and negative values. In this case, when we multiply $\mathsf{F}(s_t, a_t)$ by an heuristic, we can't be sure if the action importance will increase or decrease.

4. The methods "Heuristic from X"

One of the main questions of this paper is how to find out, in a initial learning state, the policy which must be used, speeding it up. From the class of HAL algorithms and the analysis of the heuristic function done in previous section, this question means how to define the heuristic function.

In this paper, we defined the preliminary situation as the one which takes a small percent of the necessary time for the system to converge (10total time, for instance) and corresponds to the phase where the learning process occurs in a faster way. The definition of a preliminary situation depends on the domain of the system application. For instance, in the domain of the robotic navigation, we can extract an useful heuristic from the moment when the robot is receiving environment reinforcements: after hitting a wall, use a heuristic the policy which leads the robot away from it. The second possible hypothesis in the beginning of this paper is that there's a large amount of methods and algorithms which can be used to define the heuristic function. These methods can be divided in two classes:

- based on the cases constructed previously, these algorithms work in two phases: the first one, which learns cases and the second one, which reuses cases previously analyzed. The main problem of these methods is to find features of the system which allow to index a database and which may be extracted from an initial situation.
- without a previous database (*on-line*): these methods may use self knowledge before of the domain to estimate the heuristic, or do it from just observing an execution of the system.

As a general rule, the methods "Heuristic from X" work in two stages. The first one, which withdraws information about the structure from the estimate of the domain and

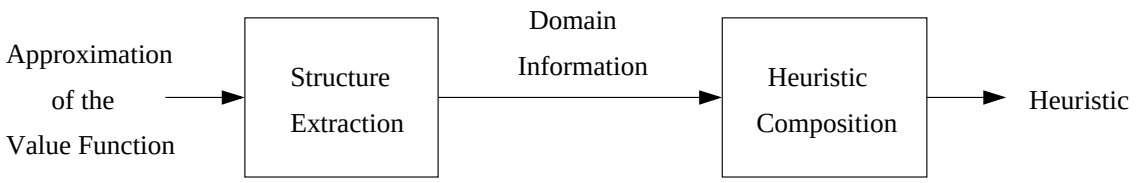

Figura 2: General method plan "Heuristic from X".

the second one, which finds the heuristic for the policy in an on-line way from a database - using the information extracted from F. These stages were called *Structure Extraction* and *Heuristic Composition*, respectively (see figure 2).

As an example of a method based in cases, a very promising method is "Heuristic from V differences", which uses the gradient of the value function to find, in a base of cases constructed previously, a policy which may be used as heuristic to a new problem. [6].

A possible *on-line* method – which uses no database – is called "Heuristic from Negative Reinforcements". In it, an auxiliary memory called $R^-(s_t, a_t)$ stores the negative reinforcements received in each state-action pair. In an environment where there are walls, and which a robot receives negative reinforcement each time it hits one of them, the R^- table indicates the map of a static environment in a short time. From this model we can estimate a heuristic which always leads the robot outside the rooms, moves away from the obstacles and corners, etc.

Finally, the method definition "Heuristic from X" depends on the designer knowledge about a domain.

5. The *Heuristically Accelerated Q–Learning* algorithm

For being the most popular algorithm and having a large amount of data in literature for the fulfillment of a comparative evaluation, the first algorithm of HAL class to be implemented is an extension of the *Q–Learning* algorithm([Watkins, 1989]). This new algorithm is named *Heuristically Accelerated Q–Learning* algorithm – HAQL algorithm.

For its implementation, it's necessary to define the rule of the state transition and the method to be used to update the heuristic. The rule of state transition used is a change of the $\epsilon - Greedy$ rule standard of the *Q–Learning* which contains the heuristic function as a simple summation (with $\beta = 1$) to the value of the function value-function. Thus, the rule of state transition in the HAQL is given by:

$$\pi(s_t) = \begin{cases} \arg\max_{a_t} \left[\hat{Q}(s_t, a_t) + \xi H_t(s_t, a_t)\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise,} \end{cases} \tag{13}$$

The value of the heuristic used in HAQL is defined by instancing the equation 12. The used heuristic is defined as:

$$H(s_t, a_t) = \begin{cases} \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + \eta & \text{if } a_t = \pi^H(s_t), \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

The convergence of this algorithm is guaranteed by 2. However, the upper limit for the error can be defined better.

[6]This method is based on the work of[Drummond, 2002].

Lemma 1 *When using the algorithm HAQL for the solution of a deterministic MDP with a set of finite states and actions, limited rewards* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, discount factor* γ *the way* $0 \leq \gamma < 1$*, the maximum value which* $Q(s_t, a_t)$ *may reach is equal to* $r_{max}/(1-\gamma)$*.*

Proof: From the equation which defines the cumulative value discounted from the Model of the Infinite Horizon and the definition of the Value-Action Q, we have:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \tag{15}$$

$$\begin{aligned} Q^*(s_t, a_t) &= r_t + \gamma V^*(s_{t+1}) \\ &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i}. \end{aligned} \tag{16}$$

where r_{t+i} is the reinforcement sequence received from the s_t state, using the π policy in a repeated way to select actions and γ is the discount factor, with $0 \leq \gamma < 1$.

Admitting, at best, all received reinforcements $r_{t+i} = r_{max}$ in all steps, we have:

$$\begin{aligned} \max Q(s_t, a_t) = &\ r_{max} + \gamma r_{max} + \gamma^2 r_{max} + \ldots \\ = &\ \sum_{i=0}^{\infty} \gamma^i r_{max} \end{aligned} \tag{17}$$

Finally, on the limit when $n \to \infty$

$$\begin{aligned} \max Q(s_t, a_t) = &\ \lim_{n\to\infty} \sum_{i=0}^{n-1} \gamma^i r_{max} \\ = &\ \frac{r_{max}}{1-\gamma}\ \square \end{aligned} \tag{18}$$

In case the positive reinforcement is attached only when coming to the final state, $r_t \leq r_{max}$ and there are no reinforcements in $t \geq t+1$, we conclude that $\forall (s_t, a_t), \max Q(s_t, a_t) \leq r_{max}$.

Lemma 2 *When using the algorithm HAQL for the solution of a deterministic MDP with a group of finite states and actions, limited rewards* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, discount factor* γ *the way* $0 \leq \gamma < 1$*, the minimum value which* $Q(s_t, a_t)$ *may admit is equal to* $r_{min}/(1-\gamma)$*.*

Proof: Admitting, at worst, all received reinforcements $r_{t+i} = r_{min}$ in all steps, excluding the last one, which receives the maximum reinforcement to achieve its goal, we can conclude that:

$$\begin{aligned} \min Q(s_t, a_t) = &\ r_{min} + \gamma r_{min} + \gamma^2 r_{min} + \ldots + \gamma^{n-1} r_{min} + \gamma^n r_{max} \\ = &\ \sum_{i=0}^{n-1} \gamma^i r_{min} + \gamma^n r_{max} \end{aligned} \tag{19}$$

On the limit when $n \to \infty$

$$\begin{aligned} \min Q(s_t, a_t) = &\ \lim_{n\to\infty} \left[\sum_{i=0}^{n-1} \gamma^i r_{min} + \gamma^n r_{max} \right] \\ = &\ r_{min} \left[\lim_{n\to\infty} \sum_{i=0}^{n-1} \gamma^i \right] \\ = &\ \frac{r_{min}}{1-\gamma}\ \square \end{aligned} \tag{20}$$

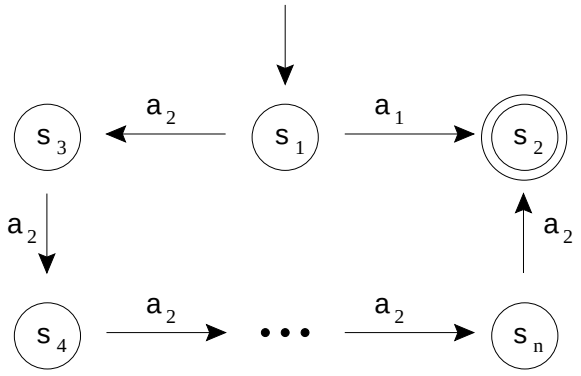

Figura 3: The s_1 state has both maximum values and minimum for the function value-action Q.

Theorem 4 *When using the algorithm HAQL for the solution of a deterministic MDP with a group of finite states and actions, limited rewards* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, discount factor* γ *the way* $0 \leq \gamma < 1$*, the maximum error caused by the use of a heuristic is limited upper by*

$$L_H(s_t) \leq \xi \left[\frac{r_{max} - r_{min}}{1-\gamma} + \eta\right]. \tag{21}$$

Proof: From the equation (14), it's possible to derive:

$$\begin{aligned} h_{min} &= 0 && \text{when } a_t \neq \pi^H(s_t), \text{ e} \\ h_{max} &= \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + \eta && \text{when } a_t = \pi^H(s_t). \end{aligned} \tag{22}$$

The value h_{max} occurs when both $\max Q(s_t, a_t)$ and $\min Q(s_t, a_t),\ \forall s_t \in S, a_t \in A$ are found under the same s_t state. In this case

$$h_{max} = \frac{r_{max}}{1-\gamma} - \frac{r_{min}}{1-\gamma} + \eta. \tag{23}$$

Substituting h_{max} and h_{min} on the result of theorem 3, we have:

$$\begin{aligned} L_H(s_t) &\leq \xi\ [h^{\beta}_{max} - h^{\beta}_{min}] \\ &\leq \xi\ \left[\frac{r_{max}}{1-\gamma} - \frac{r_{min}}{1-\gamma} + \eta - 0\right] \\ &\leq \xi\ \left[\frac{r_{max} - r_{min}}{1-\gamma} + \eta\right].\square \end{aligned} \tag{24}$$

Figure 3 presents an example of configuration where both $\max Q(s_t, a_t)$ and $\min Q(s_t, a_t)$ are found under the same s_1 state. In it, s_2 state is the final state; moving to s_2 generates a r_{max} reward and any other movement generates a r_{min} reward. In this s_1 state there is a chance of occurring *deadlocks*, because the system can't reduce the value of Q to eliminate the influence of a bad heuristic. The system will only converge after a δ time without the influence of the heuristic enough for the error $L_H(s_t)$ to be corrected by the learning process.

The complete HAQL algorithm is presented on table 2. We may notice that the only changes which refer to the use of the heuristic function for the choice of the action to be executed and the existence of a step of updating the function $H_t(s_t, a_t)$.

Although the function $H_t(s_t, a_t)$ can be derived using any method "Heuristic from X", a good method increases the speedup and generality of this algorithm. On next section the method "Heuristic from X" used in this paper is presented.

Tabela 2: The HAQL algorithm.

Initialize $Q(s, a)$.
Repeat:
Visit the s state.
Select an action a using the rule of state transition.
Receive the reinforcement $r(s, a)$ and notice the next state s'.
Update the values of $H_t(s, a)$ using a method "$\mathcal{H}$ from X".
Update the values of $Q(s, a)$ according to the rule of updating
$Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$.
Update the $s \leftarrow s'$ state.
Until some stop criteria is reached.

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

6. The method "Heuristic from Policy"

Along with HAQL a method "Heuristic from X" named "Heuristic from policy" ($\mathcal{H}$-de-π) was used. This method is composed of two phases: the first one extracts information about the structure of the environment from policy, and the second one makes the heuristic for the policy.

The first phase interactively builds the model of the environment: from the policy $\pi_t(s_t)$ in a time t, we use the algorithm of Dynamic Programming called Policy Iteration [Ribeiro, 2002] to calculate the function value $V_t^{\pi}(s_t)$ in the time t. This is done because the policy converges faster than the value function, and because of this it generates information of better quality than the direct use of $V_t s_t$.

From V_t^{π}, the gradient of the value function ∇V_t^{π} is calculated. In case of a two dimensional environment (as in the robotic navigation domain), this step corresponds to extract the edges which mark the places where there's a big variation on the value of $V_t^{\pi}(s_t)$, indicating that some feature blocks the execution of one or more actions. In case of robotic navigation, the edges can indicate the walls where the robot won't pass. Finally, the edges matrix is binarized, using a limiarization algorithm. The resulting matrix corresponds to the map of the environment.

From the model of the environment, the second phase called *Heuristic Backpropagation* makes the heuristic.

It propagates from a final state, the correct policies which lead to that state. For instance, when coming to the terminal state, we define the heuristic as composed by the actions which lead from immediately previous states, to this terminal state. In a following iteration, this heuristic is propagated to the predecessors of the states which already have a defined heuristic.

Theorem 5 *For a deterministic MDP which model is known, the Heuristic Backpropagation algorithm generates a optimal policy.*

Proof: As this algorithm is a simple application of the Dynamic Programming algorithm [Bertsekas, 1987], the theorem of Bellman itself proves this statement. □

The Heuristic Backpropagation is an algorithm very similar to the Dynamic Programming algorithm [Bertsekas, 1987]. 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

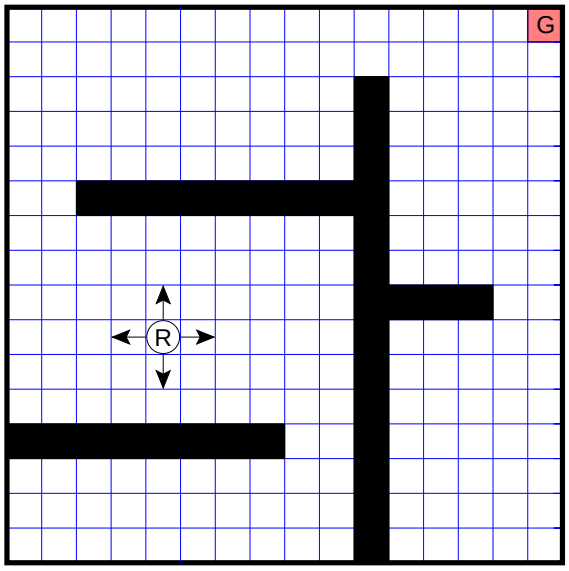

Figura 4: room with walls (represented by dark lines) discretized in a grid of states (represented by softer lines).

mapping, the model of the environment is gradually done. n this case, the backpropagation can be done only on the parts of the environment which are already mapped.

The results of complete implementation of this algorithm will be presented in the next section.

7. Experiments in the domain of mobile robots

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 domain where a mobile robot can move in four directions (North, South, East and West) was used in an environment with walls (figure 4). The domain is discretized in a grid with N x M positions to a robot, which can perform four actions: N, S, E , W. The walls are represented by states for which the robot can't move. This domain is well-known, having been used in the experiments of [Drummond, 2002] e [Foster and Dayan, 2002].

It is not difficult to find the optimal policy to this environment. Figure 5 presents the result of the algorithm of Policy Iteration [Kaelbling et al., 1996, Bertsekas, 1987], which solved the problem in 38 iterations.

Two experiments were done using the HAQL with Heuristic from Policy in this domain: navigation with repositioning goal and navigation in a new environment. In the former, the robot learns to reach the environment of the figure 5 and, after a certain time, the goal is moved to other position.

Figure 6 shows the map of the environment using the method of structure extraction described in section 6. Figure 7 shows that the policy made from this map. We can notice that neither the map nor the policy are perfect, but in spite of this, they produce good results.

The result (figures 8, 9 and 10) shows that, after the goal repositioning (which happens in the 5000^th iteration), the *Q–Learning* has to relearn all the policy, while the HAQL converges in only one step. This happens because, as the environment is already known, the HAQL just makes the heuristic to the new target position. Also, when the *Q–*

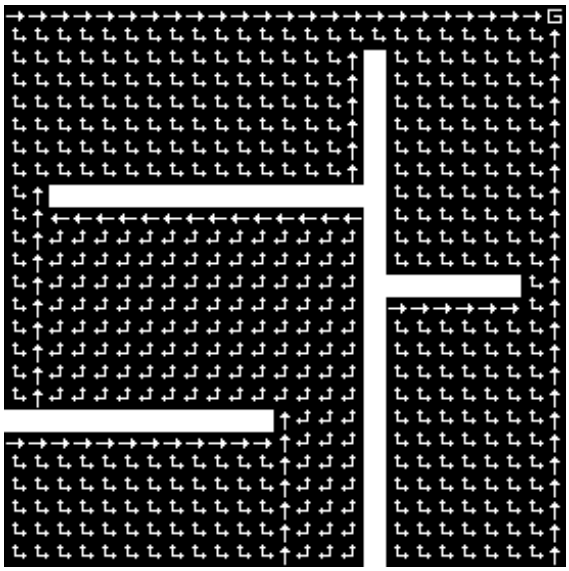

Figura 5: Optimal policy for a mobile robot in an environment with 25 x 25 states and some walls. Double arrows mean that, in a similar state, it makes no difference which one of the two actions to take.

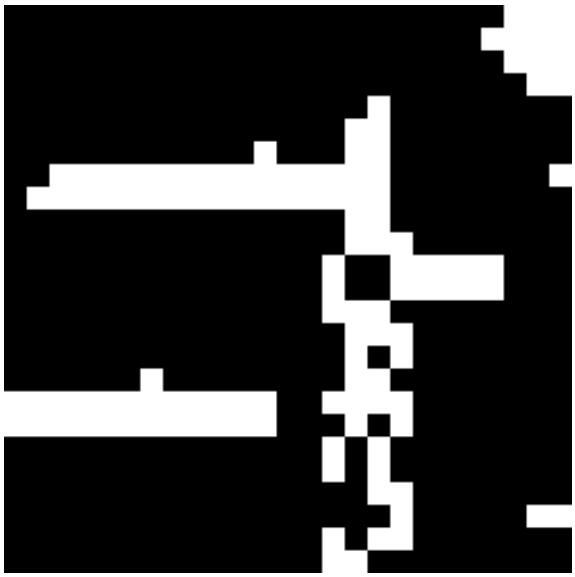

Figura 6: Environment found for the goal repositioning problem to figure 5, using the method described in section 6.

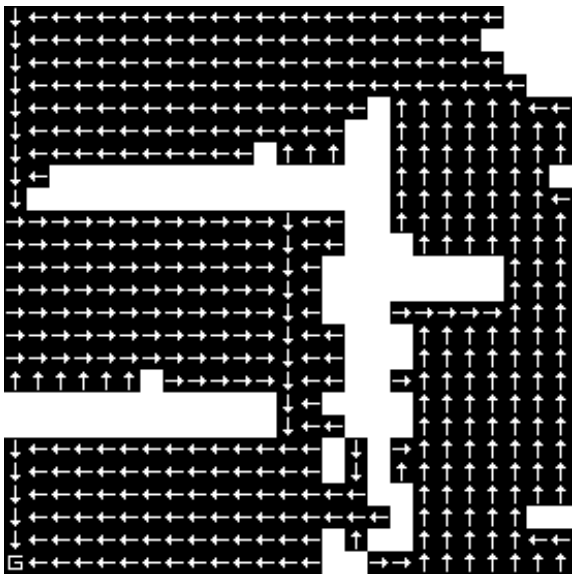

Figura 7: heuristic implemented for the goal repositioning problem using the method described on section 6.

Learning is used without the reinitialization of the table Q, the system takes about 490506 steps, but if the reinitialization is done, it takes only 7121.

The second experiment takes the Heuristic from Policy method to, on the 100^th iteration, find the map of the environment and speedup the learning process. The result (figure 11) shows that, while the *Q–Learning* learns the policy, the HAQL converges to the optimal policy after the speeding up. It's interesting to notice that in the moment of speeding up, there's a decrease on the agent's performance. This is because the map of the environment isn't perfect and some actions pointed by the heuristics as good ones aren't really. But as the agent keeps learning, these actions are soon ignored and the final result is a speedup almost immediate.

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 parameters used in both *Q–Learning* and HAQL were the same: learning rate $\alpha = 0,1$, $\gamma = 0,99$ and the exploitation rate of 0.9. The reinforcement used were: 10 when the robot reaches the goal state and -1 when it hits a wall and the presented results are a mean of 100 ages. In these similar states the Policy Iteration algorithm took 811 seconds to find the solution presented in figure 5.

8. Conclusion and Future Works

The preliminary results obtained indicate that the approach by the class of algorithms Heuristic Accelerated Learning - HAL are promising. The use of the *Q–Learning* algorithm accelerated by Policies – HAQL + $\mathcal{H}$-de-π – presented good results for the domain of mobile robots.

The biggest problems found refer to the algorithm proposed for the extraction of the problem structure. First, the necessary time for the construction of the map of the environment from the policy is very high (about 45 seconds) and second, there's the need to know the transition probabilities to solve the linear equations system using dynamic programming. Thus, this method wasn't regarded as suitable and other options are being studied.

Among the future works which must be done for a better evaluation of this algo-

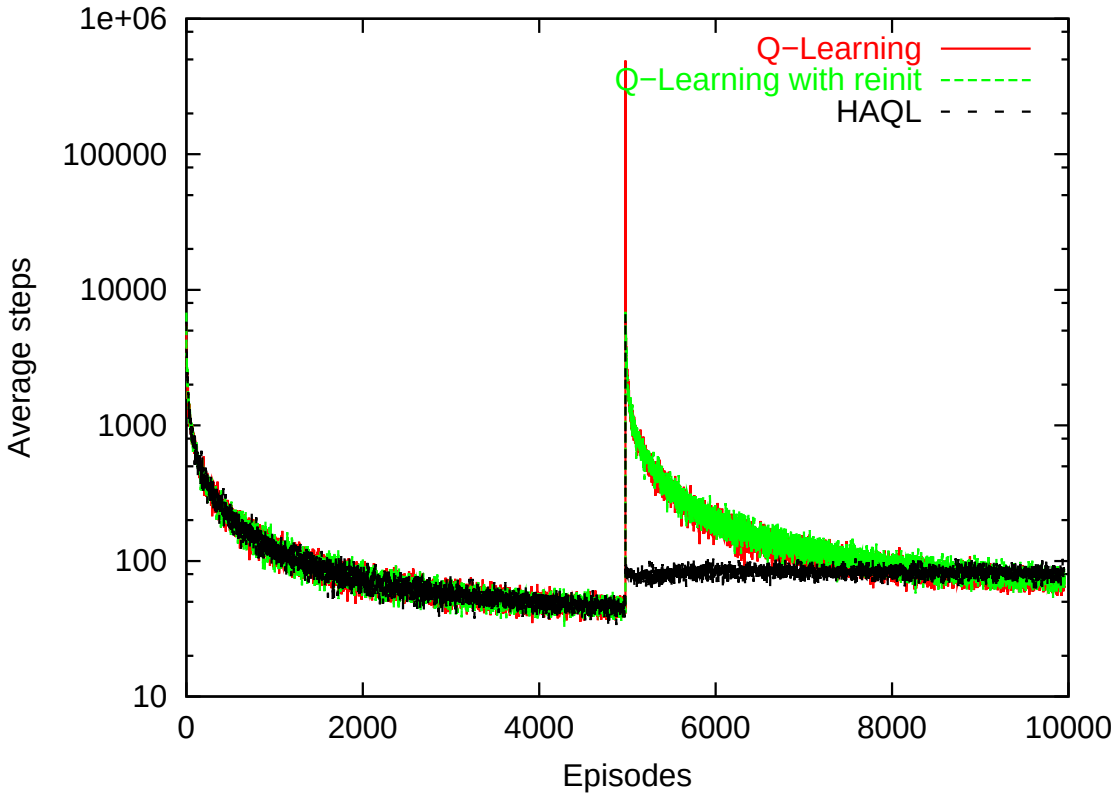

Figura 8: Result for the goal repositioning of 5000nd iteration (log y).

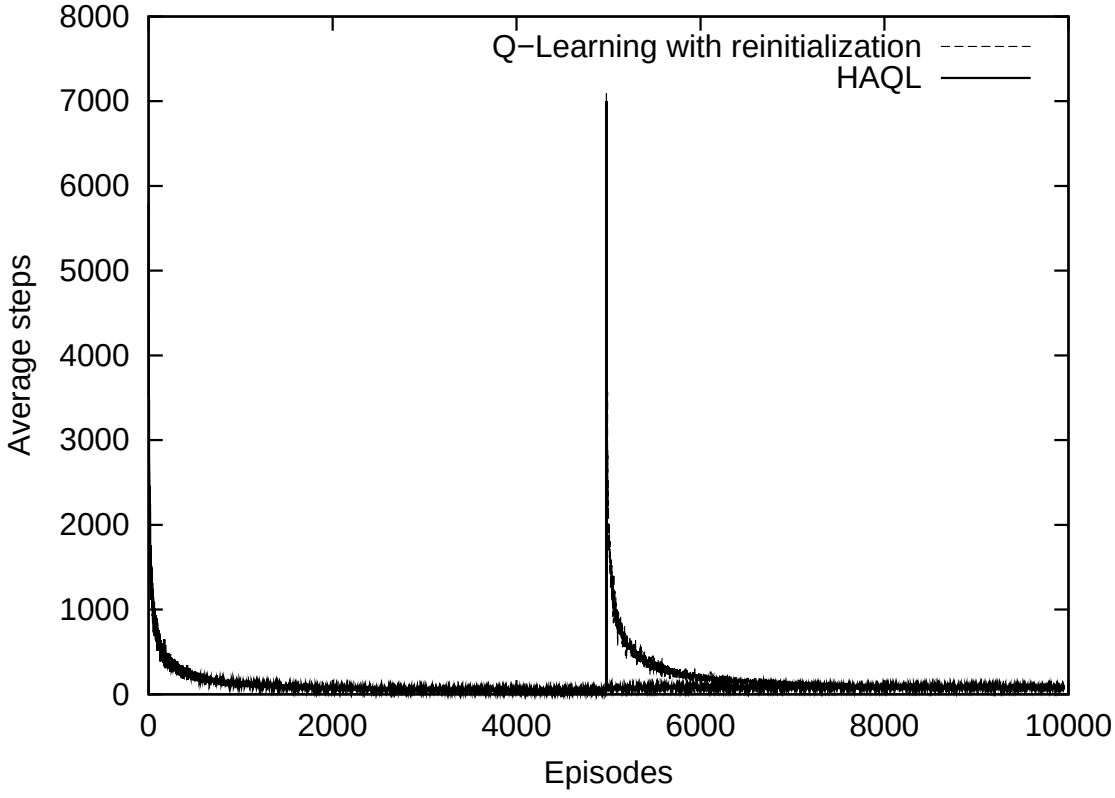

Figura 9: Result for the goal repositioning of 5000th iteration.

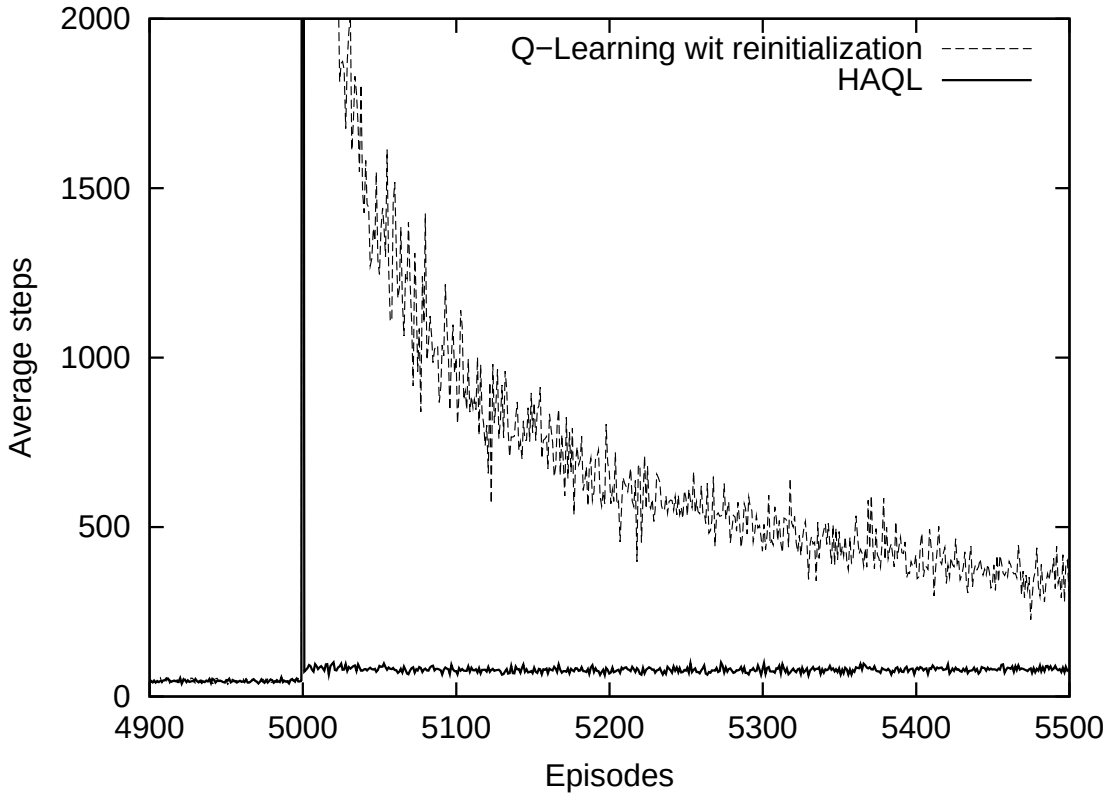

Figura 10: Result for goal repositioning on 5000th iteration.

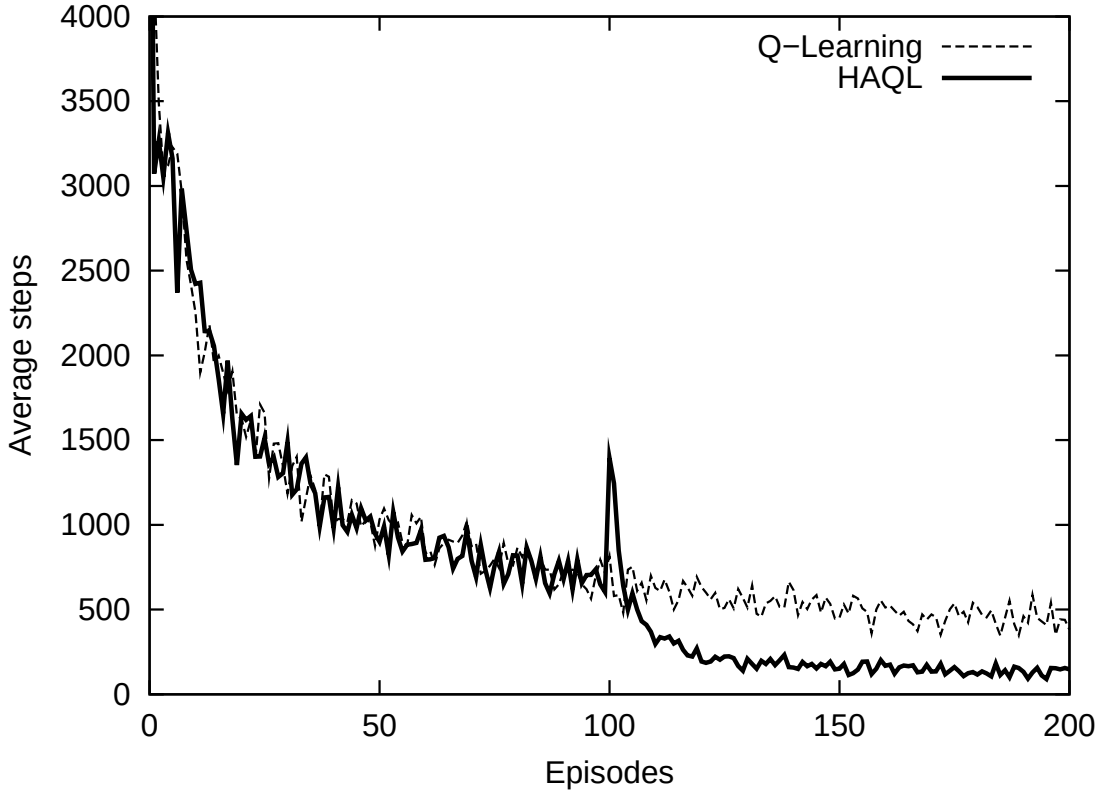

Figura 11: Result for the speedup on 100th iteration.

rithm, we have:

1. study with a deeper knowledge about the behavior of the heuristic function. Especially, what is its relation with the estimate function $\mathsf{F}(s_t, a_t)$ used, what allows to abstract which values of H to use for a certain problem.
2. validate the HAQL. To do so, there's the need to study the reutilization of knowledge learned before. For instance, on the simulation of a mobile robot, to reuse in other environments the policy which was learned in a certain configuration, there's the need to compose a new solution from parts of existing solutions. Finally, to compare this approach to the paper of [Drummond, 2002], using the same environments proposed by him.
3. Apply the algorithm HAQL also in the domain of the car on the mountain. This domain, very well-known in the control area, studies the control of a car which must stop at the top of a mountain, from its position and speed. Its feature is nonlinearity, inexistence of rooms or straight obstacles, which generates a value function where the borderlines are curves. It was studied by [Drummond, 2002] and [Munos and Moore, 2002].
4. Study other methods "Heuristic from X" which use a base of cases or not.

Referências

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

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

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

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

Gambardella, L. and Dorigo, M. (1995). 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.

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

Mitchell, T. (1997). *Machine Learning*. McGraw Hill, New York.

Munos, R. and Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning*, 49(2/3):291–323.

Nalwa, V. S. (1993). *A guided tour of computer vision*. Addison-Wesley, Reading, MA.

Ribeiro, C. H. C. (2002). Reinforcement learning agents. *Artificial Intelligence Review*, 17:223–250.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

Singh, S. P. (1994). *Learning to solve Markovian Decision Processes*. PhD thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1).

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

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge.