1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

On the relation between Ant Colony Optimization and Heuristically Accelerated Reinforcement Learning *

Reinaldo A. C. Bianchi
Centro Universitário FEI
São Bernardo do Campo, Brazil.
rbianchi@fei.edu.br

Carlos H. C. Ribeiro
Technological Institute of Aeronautics
Computer Science Division
São José dos Campos, Brazil.
carlos@comp.ita.br

Anna H. R. Costa
Escola Politécnica
University of São Paulo
São Paulo, Brazil.
anna.reali@poli.usp.br

Abstract

This paper has two main goals: the first is to propose a new class of Heuristically Accelerated Reinforcement Learning algorithms (HARL), the Distributed HARLs, describing one algorithm of this class, the Heuristically Accelerated Distributed Q-Learning (HADQL); and the second is to show that Ant Colony Optimization (ACO) algorithms can be seen as instances of Distributed HARLs algorithms. In particular, this paper shows that the Ant Colony System (ACS) algorithm can be interpreted as a particular case of the HADQL algorithm. This interpretation is very attractive, as many of the conclusions obtained for RL algorithms remain valid for Distributed HARL algorithms, such as the guarantee of convergence to equilibrium. In order to better evaluate the proposal, we compared the performances of the Distributed Q-Learning, the HADQL and the ACS algorithms in the Traveling Salesman Problem domain. The results show that HADQL and the ACS algorithm have similar performances, as it would be expected from the hypothesis that they are, in fact, instances of the same class of algorithms.

1 Introduction

In the last few years several researchers noticed the similarity between Ant Colony Optimization (ACO) and Reinforcement Learning (RL) [Stützle and Dorigo, 2002; Dorigo and Blum, 2005]. Despite having different inspirational sources – ACO is inspired in the foraging behavior of real ants, while RL is based on Optimal Control Theory –, they have several characteristics in common, such as the use of Markov Decision Process as a way to formulate the problem and provide convergence proofs for the algorithms [Stützle and Dorigo, 2002], and the similarity between the action-value function in RL and the pheromone in ACO, among other aspects.

The major differences between ACO and RL are that the first is a distributed approach, with several agents working to find a solution to a given problem and that ACO makes use of a heuristic evaluation of which moves are better. This last difference – the use of heuristics by the ACO – made it difficult to completely model an ACO algorithm as a RL one.

Now these differences can be addressed by using a recently proposed technique, the Heuristically Accelerated Reinforcement Learning (HARL) [Bianchi *et al.*, 2008]. This technique was proposed to speed up RL methods 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. HARL techniques are based on firm theoretical foundations, allowing many of the conclusions obtained for RL to remain valid, such as the guarantee of convergence to an optimal solution in the limit.

This paper presents two contributions: the first one is the proposal of a new class of HARL algorithms, the Distributed HARL (HADRL), and the description and implementation of one algorithm of this class, the Heuristically Accelerated Distributed Q-Learning (HADQL), which extends the Distributed Q-learning (DQL) algorithm proposed by Mariano and Morales [2001]. The second contribution is a demonstration that ACO algorithms can be seen as instances of Distributed HARLs algorithms. In particular, the paper shows that the Ant Colony System (ACS) algorithm [Dorigo and Gambardella, 1997] can be considered a particular case of the HADQL algorithm. This interpretation is very attractive, as many of the conclusions obtained for RL algorithms remain valid for Distributed HARL algorithms, such as the guarantee of convergence to equilibrium.

The domain studied herein is that of the Traveling Salesman Problem, which is used as a benchmark for testing the algorithms with the goal of evaluating HADQL and comparing it with the DQL and the ACS. Nevertheless, the technique proposed in this work is domain independent.

The remainder of this paper is organized as follows: Section 2 briefly reviews the RL problem and the Distributed Q–Learning algorithm, while Section 3 describes the HAQL approach and its solutions. Section 4 shows how to incorporate heuristics in the DQL algorithm. Section 5 presents the ACO and the ACS algorithm, and Section 6 shows how ACS can be seen as a HARL algorithm. Section 7 presents the experiments performed and shows the results obtained. Finally, Section 8 provides our conclusions and outlines future work.

*The authors would like to thanks FAPESP for supporting this project. Reinaldo Bianchi also acknowledge the support of the CNPq (Grant No. 201591/2007-3) and FAPESP (Grant No. 2009/01610-1).

2 Reinforcement Learning and the DQL algorithm

Reinforcement Learning (RL) algorithms have been applied successfully to the on-line learning of optimal control policies in Markov Decision Processes (MDPs). In RL, this policy is learned through trial-and-error interactions of the agent with its environment: on each interaction step the agent senses the current state s of the environment, chooses an action a to perform, executes this action, altering the state s of the environment, and receives a scalar reinforcement signal r (a reward or penalty).

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.

The goal of the agent in a RL problem is to learn an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maps the current state s into the most desirable action a to be performed in s. One strategy to learn the optimal policy π^* is to allow the agent to learn the evaluation function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. Each action value $Q(s, a)$ value represents the expected cost incurred by the agent when taking action a at state s and following an optimal policy thereafter.

The Q–learning algorithm [Watkins, 1989] is a well-know RL technique that uses a strategy to learn an optimal policy π^* via learning of the action values. It iteratively approximates Q, provided the system can be modeled as an MDP, the reinforcement function is bounded, and actions are chosen so that every state-action pair is visited an infinite number of times. The Q learning update rule is:

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

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 ($0 \leq \gamma < 1$); and α, is the learning rate.

In the case of the use of the Q–Learning algorithm to solve the TSP, the state s corresponds to the city in which the agent is at a given moment, and the set of actions that an agent can execute corresponds to the set of cities in the problem, excluding the current city s.

To select an action to be executed, the Q–Learning algorithm usually considers an $\epsilon - Greedy$ strategy:

$$\pi(s) = \begin{cases} \arg\max_a \hat{Q}(s, a) & \text{if } q \leq p, \\ a_{random} & \text{otherwise} \end{cases} \quad (2)$$

where q is a random value uniformly distributed over $[0, 1]$ and p ($0 \leq p \leq 1$) is a parameter that defines the exploration/exploitation tradeoff: the larger p, the smaller is the probability of executing a random exploratory action, and a_{random} is an action randomly chosen among those available in state $\mathbf{s}_t$.

Table 1: The DQL algorithm [Mariano and Morales, 2001].

Initialize $Q(s, a)$ arbitrarily
Repeat (for n episodes):
  Repeat (for each agent i in the set of m agents):
    Initialize s, copy $Q(s, a)$ to $Qc^i(s, a)$
    Repeat (for each step of the episode):
      Select an action a, observe r, s'.
      Update the $Qc^i(s, a)$ values.
      $s \leftarrow s'$.
    Until s is terminal.
  Evaluate the m solutions.
  Assign reward to the best solution found.
  Update the global $Q(s, a)$.
Until a termination criterion is met.

Several authors have proposed distributed approaches to RL, and various forms of distributed Q–Learning were developed [Pendrith, 2000; Lauer and Riedmiller, 2000; Gu and Maddox, 1996; Mariano and Morales, 2001]. One of these is the Distributed Q-learning algorithm (DQL) proposed by Mariano and Morales [2001], which is a generalization of the Q-learning algorithm where, instead of a single agent, several independent agents are used to learn a single policy.

In Mariano and Morales' DQL, in addition to the global $Q(s, a)$ function, each agent i keeps a temporary copy of Q, the $Qc_i(s, a)$ that is used to decide which action to perform, following an ϵ-greedy policy. Every time an action is performed, $Qc_i(s, a)$ is updated according to the Q-Learning update rule (Equation 1).

The DQL agents explore different options in a common environment and when all agents have completed a solution, their solutions are evaluated, and the global $Q(s, a)$ table is updated: the best solution (the shortest route made by all agents in the TSP) is updated using Equation (1), and receiving a reward r. The DQL algorithm is presented in table 1.

As DQL does not change the update rules of the Q–Learning algorithm, the same proofs of convergence used for the standard Q–Learning remains valid for it [Mariano and Morales, 2001]. Different Distributed Q-Learning algorithms, proposed by other authors, also hold convergence proofs [Lauer and Riedmiller, 2000].

3 Heuristic Accelerated Reinforcement Learning

Formally, a Heuristically Accelerated Reinforcement Learning (HARL) algorithm 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 the action a when visiting state s. The heuristic function is strongly associated with the policy: every heuristic indicates that an action must be taken regardless of others.

The heuristic function is an action policy modifier which does not interfere with the standard bootstrap-like update

mechanism of RL algorithms. A possible strategy for action choice is an $\epsilon - greedy$ mechanism where a heuristic mechanism formalized as a function $H(s,a)$ is considered, thus:

$$\pi(s) = \begin{cases} \arg\max_a \left[\mathsf{F}(s,a) \bowtie \xi H(s,a)^\beta\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise} \end{cases} \quad (3)$$

where:

- $\mathcal{F} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is an estimate of a value function that defines the expected cumulative reward. If $\mathsf{F}(s,a) \equiv \hat{Q}(s,a)$ we have an algorithm similar to standard *Q–Learning*.
- $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the heuristic function that plays a role in the action choice. $H(s,a)$ defines the importance of executing action s in state s.
- $\bowtie$ is a function that operates on real numbers and produces a value from an ordered set which supports a maximization operation.
- ξ and β are design parameters that control the influence of the heuristic function.
- q is a parameter that defines the exploration/exploitation tradeoff.
- a_{random} is an action randomly chosen among those available in state s.

In general, the value of $H(\mathbf{s},a)$ must be larger than the variation among the values of $\mathsf{F}(\mathbf{s}_a)$ for a given $\mathbf{s} \in S$, so that it can influence the action choice. On the other hand, it must be as small as possible in order to minimize the error. If $\bowtie$ is a sum and $\xi = \beta = 1$, a heuristic can be defined as:

$$H(\mathbf{s},a) = \begin{cases} \max_a \left[\mathsf{F}(\mathbf{s},a)\right] - \mathsf{F}(\mathbf{s},a) + \eta & \text{if } a = \pi^H(\mathbf{s}), \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

where η is a small value and $\pi^H(\mathbf{s})$ is a heuristic obtained using an appropriate method.

For instance, let [1.0 1.1 1.2 0.9] be the values of $\mathsf{F}(\mathbf{s},a)$ for four possible actions $[a_1\ a_2\ a_3\ a_4]$ for a given state $\mathbf{s}_t$. If the desired action is the first one (a_1), we can use $\eta = 0.01$, resulting in $H(\mathbf{s},a_1) = 0.21$ and zero for the other actions (see Figure 1). The heuristic can be defined similarly for other definitions of $\bowtie$ and values of ξ and β.

Convergence of the first HARL algorithm — Heuristically Accelerated Q–Learning (HAQL) — is presented in Bianchi *et al*. [2008], together with the definition of an upper bound for the error in the estimation of Q. The same authors investigated the use of HARL in multiagent domain, proposing a multiagent HARL algorithm – the Heuristically Accelerated Minimax-Q [Bianchi *et al*., 2007] – and testing it in a simplified simulator for the robot soccer domain.

4 The HADQL algorithm

The Heuristically Accelerated Distributed Q–Learning algorithm is a HARL algorithm that extends the DQL algorithm by making use of an heuristic function in the action choice

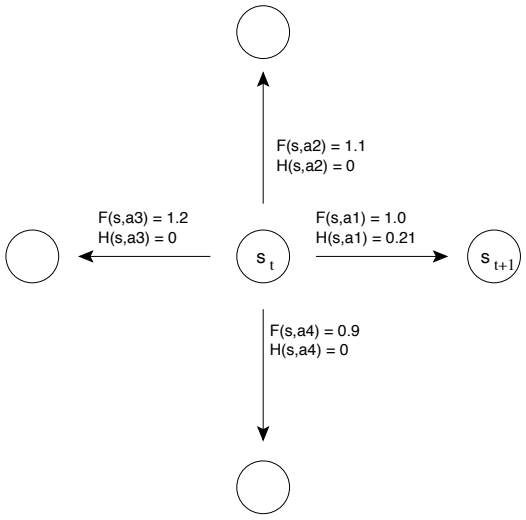

Figure 1: Suppose a state s_t and a desired state s_{t+1}. The value of $H(s_t,a_1)$ for the action that leads to s_{t+1} is 0.21, and zero for the other actions.

rule defined in Equation (3), where $\mathcal{F} = Q$, the $\bowtie$ operator is the sum and $\beta = 1$:

$$\pi(s) = \begin{cases} \arg\max_a \left[\hat{Q}(s,a) + \xi H(s,a)\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise,} \end{cases} \quad (5)$$

where all variables are defined as in Equation (3). The value of the heuristic can be defined by instantiating Equation 4:

$$H(s,a) = \begin{cases} \max_i \hat{Q}(s,i) - \hat{Q}(s,a) + \eta \text{ if } a = \pi^H(s), \\ 0 \text{ otherwise.} \end{cases} \quad (6)$$

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

As the heuristic is used only in the choice of the action to be taken, the DQL operation is not modified (i.e., updates of the function Q are as in Q–learning), and it thus allows that many of the conclusions obtained for DQL remain valid for HADQL. The HADQL algorithm is presented in table 2.

Theorem 1. *Consider a HADQL system learning in a deterministic MDP, with finite sets of states and actions, bounded rewards* $(\exists c \in \Re; (\forall s,a), |r(s,a)| < c)$*, discount factor* γ *such that* $0 \leq \gamma < 1$ *and where the values used on the heuristic function are bounded by* $(\forall s,a)\ h_{min} \leq H(s,a) \leq h_{max}$*. For this algorithm, 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 HADQL, the update of the value function approximation does not depend explicitly on the value of the heuristic. The necessary conditions for the convergence of DQL that could be affected with the use of the heuristic algorithm

Table 2: The HADQL algorithm

Initialize $Q(s,a)$ and $H(s,a)$ arbitrarily
Repeat (for n episodes):
  Repeat (for each agent i in the set of m agents):
    Initialize s, copy $Q(s,a)$ to $Qc^i(s,a)$
    Repeat (for each step of the episode):
      Compute the value of $H(s,a)$ using Eq. 6.
      Select an action a using Eq. 5.
      Observe r, s'.
      Update the $Qc^i(s,a)$ values.
      $s \leftarrow s'$.
    Until s is terminal.
  Evaluate the m solutions.
  Assign reward to the best solution found.
  Update the global $Q(s,a)$.
Until a termination criterion is met.

HADQL are the ones that depend on the choice of the action. Of the conditions presented in Littman and Szepesvári [1996]; Mitchell [1997], the only one that depends on the action choice is the necessity of infinite visitation to each pair state-action. As equation 5 considers an exploration strategy $\epsilon-$ *greedy* regardless of the fact that the value function is influenced by the heuristic, the infinite visitation condition is guaranteed and the algorithm converges. □

The condition of infinite visitation of each state-action pair can be considered valid in practice also by using visitation strategies such as Boltzmann exploration [Kaelbling *et al.*, 1996], intercalating steps where the algorithm makes alternate use of the heuristic and exploration, or using the heuristic during a period of time, smaller than the total learning time.

The domain studied in this work is that of the Traveling Salesman Problem (TSP), which consists in, given a number n of cities $C = c_1, c_2, \ldots c_n$ and the distance $d_{i,j}$ between them, to find the shortest route that visits each city in C at least once and then returns to the starting city.

To compute an heuristic function for the TSP problem, we were inspired by the Nearest Neighbor Heuristic used in several works [Russell and Norvig, 1995]. This heuristic states that the agent starts at some city and from there it visits the nearest city that was not visited so far. Using this rule, a simple heuristic that indicates to which city an agent must move can be defined as the inverse of the distance $d_{i,j}$ between the cities i and j times a constant η:

$$H(s,a) = \frac{\eta}{d_{i,j}}, \tag{7}$$

where s is the current state (i.e., the current city c_i) and a is the action that takes the agent to the city c_j.

The problem with this heuristic is that it does not take into account that the agents must not visit two times the same city. Therefore, the action chosen as the one to be done (the heuristic policy) is the one that moves the agent to the nearest city that was not visited yet. To put this idea in the framework of equation 6, and taking into account the cities that were already visited, the heuristic function becomes:

$$H(s,a) = \begin{cases} \max\limits_{x} \hat{Q}(s,x) - \hat{Q}(s,a) + \eta & \text{if } \delta_{i,j} = \min\limits_{y} \delta_{i,y} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

where η is a small real value (usually 1), a is the action that takes the agent from city i to j and $\delta_{i,j}$ the distances between the city c_i and the cities that have not been visited so far, defined by:

$$\delta_{i,j} = \begin{cases} d_{i,j} & \text{if } j \text{ has not been visited} \\ \infty & \text{otherwise.} \end{cases} \tag{9}$$

Finally, the agents receive only negative reinforcements, defined as minus the distance between the cities, $r = -d_{i,j}$.

5 The Ant Colony System Algorithm

Based on the social insect metaphor for solving problems, the use of Ant Colony Optimization (ACO) for solving several kinds of problems has attracted an increasing attention of the AI community [Bonabeau *et al.*, 1999, 2000; Dorigo and Blum, 2005]. The Ant Colony System (ACS) is an ACO algorithm proposed by Dorigo and Gambardella [1997] for combinatorial optimization based on the observation of ant colonies behavior.

ACS has been applied to various combinatorial optimization problems like the symmetric and Asymmetric Traveling Salesman Problems (TSP and ATSP respectively), and the quadratic assignment problem.

ACS represents the usefulness of moving to the city s when in city r in $\tau(r,s)$, called *pheromone*, which is a positive real value associated to the edge (r,s) in a graph. There is also a *heuristic* $\eta(r,s)$ associated to the edge (r,s). It represents an heuristic evaluation of which moves are better. In the TSP, $\eta(r,s)$ can be the inverse of the distance δ from r to s, $\delta(r,s)$.

An agent k positioned in city r moves to city s using the following rule, called state transition rule [Dorigo and Gambardella, 1997]:

$$s = \begin{cases} \arg\max\limits_{u \in J_k(r)} \tau(r,u) \cdot \eta(r,u)^{\beta} & if\ q \leq q_0 \\ S & otherwise \end{cases} \tag{10}$$

where:

- β is a parameter which weights the relative importance of the learned pheromone and the heuristic distance values ($\beta > 0$).
- $J_k(r)$ is the list of cities still to be visited by the ant k, where r is the current city. This list is used to constrain agents to visit cities only once.
- q is a parameter that defines the exploitation/exploration rate.
- S is a random city from the list of cities $J_k(r)$.

Ants in ACS update the values of $\tau(r,s)$ in two situations: in the local update step (applied when ants visit edges) and in the global update step (applied when ants complete the tour).

Table 3: The ACS algorithm (in the TSP Problem).

Initialize the pheromone table, the ants and the list of cities.
Repeat (for n episodes):
  Repeat (for each ant i in the set of m ants):
    Put each ant at a starting city.
    Repeat (for each step of the episode):
      Chose next city using Equation (10).
      Update list J_k of yet to be visited cities for ant k.
      Apply local update using Equation (11).
    Until (ants have a complete tour).
  Apply global update using Equation (12).

The ACS local update rule is:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\Delta\tau(r,s) \qquad (11)$$

where $0 < \rho < 1$ is a the learning rate, and $\Delta\tau(r,s) = \gamma \cdot \max_{z\in J_k(s)} \tau(s,z)$.

The ACS global update rule is:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau(r,s) \qquad (12)$$

where α is the pheromone decay parameter, and $\Delta\tau(r,s)$ is the inverse of the length of the best tour, given only to the tour done by the best agent – only the edges belonging to the best tour will receive more pheromones.

Thus, the pheromone updating formulas aims at placing a greater amount of pheromone on the shortest tours, achieving this by simulating the addition of new pheromone deposited by ants and evaporation. The ACS algorithm is presented in table 3.

6 ACS as HARL

Now that the HADQL algorithm is defined, we proceed in explaining how ACS can be seen as an HARL algorithm:

- The Pheromone is the ACS counterpart of DQL Q-values, where the city r in which the ant is at a defined moment corresponds to the state s, and the city s the ant should move to, corresponds to the action a to be taken: $\tau(r,s) = \hat{Q}(s,a)$;
- The heuristic $\eta(r,u)$ corresponds to the heuristic function $H(s,a)$;
- The state transition rule (Eq. 10) is the same as the one used by HARL algorithms (Eq. 3), with the $\bowtie$ operator being the multiplication, $\xi = 1$ and β having the same function;
- The list $J_k(r)$ of cities still to be visited by the ant, which causes problems for the convergence proof of ACS (because it cannot be defined as a MDP - see Koenig and Simmons [1996]), can be encoded as an heuristic, as in Equation 8;
- The local update step (Eq. 11) of $\tau(r,s)$ is made in the same way as the updating of Qc in DQL (without giving a reinforcement),
- The global update step (Eq. 12) in ACS corresponds to attributing the reinforcement to the best solution, which is made in the updating of the global Q of DQL, with the pheromone decay parameter α being equal to the learning factor and $\Delta\tau(r,s)$ corresponding to a delayed reward.

ACS and HADQL as proposed in this work differ by the fact that HADQL partial updates are performed over copies of the Q-table, and that HADQL updates the best solution at the same time it provides the reinforcement (that is, the last step of HADQL corresponds to executing both a local and a global ACS update at the same time). According to Mariano and Morales [2001], this avoids multiple updates of the same Q-table, resulting in updating only relevant solutions and allowing faster convergence.

Finally, by modeling ACS as a HARL, it is possible to show that ACS also converges to equilibrium in the limit.

Theorem 2. *Consider ACS in a deterministic MDP, with finite sets of states and actions (a finite number of cities to visit), bounded rewards* $(\exists c \in \Re; (\forall r,s), |\Delta\tau(r,s)| < c)$, *learning rate* ρ *and pheromone decay parameter* α *with values between* 0 *and* 1 *and where the values used on the heuristic are bounded by* $(\forall r,s)\ \eta_{min} \leq \eta(r,s) \leq \eta_{max}$. *For this algorithm, the* $\hat{\tau}$ *values will converge to* τ^*, *with probability one uniformly over all the states* $s \in \mathcal{S}$, *if each pair* (r,s) *is visited infinitely often*.

Proof. The only difference between ACS and HADQL is that ACS executes more local updates. But the effect of having more local updates is that the pair (r,s) is visited more often, assigning rewards more frequently. As HADQL converges, without executing the local updates, ACS also converges. □

Other ways by which it could be shown that ACS converges is by modeling it as a HADQL that includes cooperation among the agents, such as one based on Lauer and Riedmiller [2000], or by using a DQL that only posses one Q-table, such as in Pendrith [2000].

7 Experiments in the TSP domain

In order to evaluate the performance of the HADQL algorithm and its relation with ACS, this section compares the performances of these algorithms while solving a set of TSPs. The Distributed Q-learning (DQL) algorithm is also included in this comparison, for the sake of comparing the new algorithm with a traditional RL one.

These tests were performed using a standardized dataset, the TSPLIB [Reinelt, 1995]. This library of problems, which was used as benchmark by both Dorigo and Gambardella [1997] and Mariano and Morales [2001], offers standardized optimization problems such as the TSP, truck loading and unloading and crystallography problems. The results, which are the average of 30 training sessions with 1000 episodes, are presented in Tables 4, 6, and 5. Ten different problems were considered: the first 7 ones are TSPs, and the last 3 ones are ATSPs (Asymmetric TSPs). The number of cities in each problem ranged from 48 to 170, and is shown in the name of the problem (for example, 52 cities in the 'berlin52' problem, and so on).

Table 4 presents the best result found by DQL, ACS and HADQL after 1000 iterations (in 30 trials). It can be seen

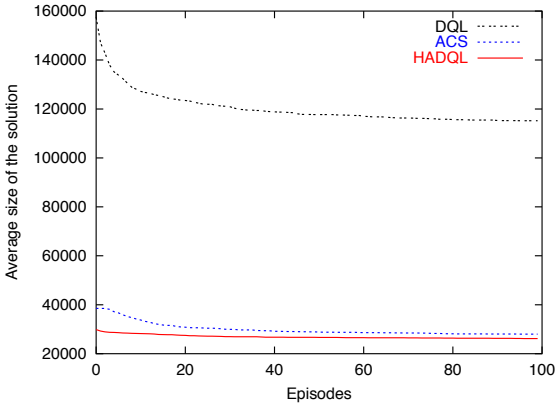

Figure 2: Comparison between the algorithms DQL, ACS and HADQL applied to the kroA100 TSP.

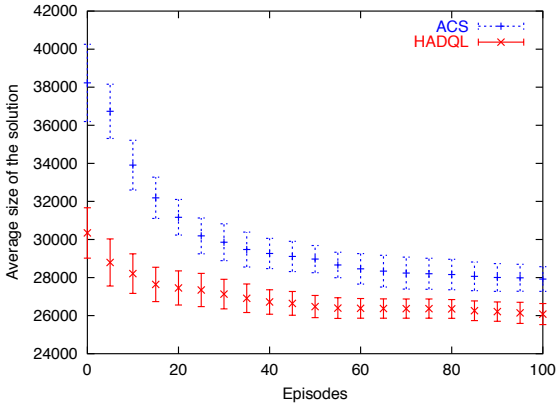

Figure 3: Comparison between the algorithms ACS and HADQL applied to the kroA100 TSP (with errorbars).

Problem	Known Solution	DQL	ACS	HADQL
berlin52	7542	15424	8689	7824
kroA100	21282	105188	25686	24492
kroB100	22141	99792	27119	23749
kroC100	20749	101293	24958	22735
kroD100	21294	101296	26299	23839
kroE100	22068	102774	26951	24267
kroA150	26524	170425	33748	32115
ry48	14422	26757	16413	15398
kro124	36230	122468	45330	42825
ftv170	2755	18226	4228	3730

Table 4: Shortest routes found by the algorithms DQL, ACS and HADQL after 1000 episodes (best of 30 trials).

Problem	DQL	ACS	HADQL
berlin52	16427 ± 540	8589 ± 139	7929 ± 61
kroA100	108687 ± 2474	26225 ± 542	25114 ± 353
kroB100	105895 ± 2949	27337 ± 582	24896 ± 463
kroC100	105756 ± 2710	25985 ± 737	23585 ± 361
kroD100	104909 ± 2293	26188 ± 533	24441 ± 274
kroE100	108098 ± 2652	26723 ± 557	25196 ± 359
kroA150	179618 ± 3397	35816 ± 998	33532 ± 603
ry48	29562 ± 1131	16285 ± 195	15538 ± 58
kro124	127911 ± 2485	46394 ± 599	43482 ± 322
ftv170	19278 ± 373	4532 ± 104	3982 ± 98

Table 5: Average size of the results found by the algorithms DQL, ACS and HADQL after 1000 episodes (average of 30 trials).

that HADQL solutions are better than the solutions from the other two algorithms for all the problems. The same occurs for the average of the best results found after 1000 episodes (Table 5). Figure 2 shows the evolution of the average of the results found by DQL, ACS and HADQL when solving the kroA100 problem, and Figure 3 shows the same results for the ACS and HADQL, with errorbars. It is possible to see in both figures that HADQL converges faster to the solution.

The average time to find these solutions, shown in Table 6, indicates that the DQL is the fastest algorithm. This occurs because DQL converges first, but to a solution of poorer quality. A comparison between mean times to find the best solution between the ACS and HADQL algorithms shows that there is no significant difference between them. It is worth noticing that small improvements may occur at any time, because both algorithms are following an $\epsilon-greedy$ exploration policy. For this reason, the variation in results is very large, which is reflected in the error measure.

Finally, Student's t–test [Spiegel, 1998] was used to verify the hypothesis that the HADQL algorithm produces better results that the ACS. This test showed that all the results of the HADQL are better than the ones obtained with ACO, with a level of confidence greater than 99,99%, a fact that shows that the small differences that exists between both algorithms is enough to make HADQL perform slightly but consistently better than ACS. The same test applied to the average time to reach the best result showed that there is no significant difference in the performance of the two algorithms.

The parameters used in the experiments were the same for the three algorithms: the learning rate is $\alpha = 0,1$, the exploration/exploitation rate is $p = 0.9$ and the discount factor $\gamma = 0.3$. The value of β in the ACS algorithm was set to 2 and the value of $\eta = 10$ in the HADQL (these parameters are identical to those used by Dorigo and Gambardella [1997] and Mariano and Morales [2001]). Values in the Q table were randomly initiated, with $0 \leq Q(s,a) \leq 1$. The experiments were programmed in C++ and executed in a AMD Athlon 2.2MHz, with 512MB of RAM in a Linux platform.

8 Conclusion

In this paper we proposed a new algorithm – HADQL, showed that ACS can be seen as a HARL algorithm and compared the two algorithms in the TSP domain. The results

Problem	DQL	ACS	HADQL
berlin52	7 ± 3	12 ± 7	11 ± 6
kroA100	37 ± 13	89 ± 50	73 ± 43
kroB100	44 ± 17	85 ± 44	88 ± 47
kroC100	51 ± 27	82 ± 48	89 ± 38
kroD100	47 ± 21	98 ± 39	74 ± 39
kroE100	48 ± 22	80 ± 43	80 ± 45
kroA150	91 ± 42	267 ± 136	294 ± 154
ry48	6 ± 3	9 ± 6	3 ± 4
kro124	62 ± 25	89 ± 42	95 ± 43
ftv170	113 ± 73	317 ± 122	333 ± 221

Table 6: Average time (in seconds) to find the best solution using the algorithms DQL, ACO and HADQL, limited to 1000 episodes.

show that HADQL and ACS have similar performances, as it would be expected from the hypothesis that they are, in fact, instances of the same class of algorithms.

Despite the similarity between the HADQL and the ACS algorithms, the results showed a small advantage for the first one. We believe that this advantage is caused by the fact that the HADQL performs partial updates over copies of the Q-table, avoiding updates without rewards, and by the fact that the reward given in both algorithms are different.

Future work include testing other forms of combining Q-values and heuristics in the action selection, and the comparison of other ACO and HARL algorithms, such as comparing the Max-Min Ant System (MMAS) [Dorigo and Blum, 2005] with the HAMMQ [Bianchi *et al*., 2007].

References

Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, and Anna H. R. Costa. Heuristic selection of actions in multiagent reinforcement learning. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'2007), Hyderabad, India, January 6-12, 2007*, pages 690–695, 2007.

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.

Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. Inspiration for optimization from social insect behaviour. *Nature 406 [6791]*, 2000.

Marco Dorigo and Christian Blum. Ant colony optimization theory: a survey. *Theor. Comput. Sci.*, 344(2-3):243–278, 2005.

Marco Dorigo and Luca M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

Pan Gu and Anthony B. Maddox. A framework for distributed reinforcement learning. In Gerhard Weiß and Sandip Sen, editors, *Adaption and Learning in Multi-Agent Systems, IJCAI'95 Workshop Proceedings, Monréal, Canada, August 21, 1995*, volume 1042 of *Lecture Notes in Computer Science*, pages 97–112, Berlin Heidelberg, Springer, 1996.

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

Sven Koenig and R. G. Simmons. The effect of representation and knowledge on goal–directed exploration with reinforcement–learning algorithms. *Machine Learning*, 22:227–250, 1996.

Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Standord, CA, USA, June 29 - July 2, 2000*, pages 535–542. Morgan Kaufmann, 2000.

Michael L. Littman and Csaba Szepesvári. A generalized reinforcement learning model: convergence and applications. In Lorenza Saitta, editor, *Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 310–318. Morgan Kaufmann, 1996.

Carlos E. Mariano and Eduardo F. Morales. DQL: A new updating strategy for reinforcement learning based on Q–learning. In Luc De Raedt and Peter A. Flach, editors, *EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5–7, 2001*, volume 2167 of *Lecture Notes in Computer Science*, pages 324–335, Berlin Heidelberg, Springer, 2001.

Tom Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

Mark D. Pendrith. Distributed reinforcement learning for a traffic engineering application. In Carles Sierra, Maria Gini, Jeffrey S. Rosenschein, editors, *Proceedings of the fourth international Conference on Autonomous agents, June 3-7, 2000, Barcelona, Catalonia, Spain*, pages 404–411. ACM, 2000.

Gerhard Reinelt. Tsplib95. Technical report, Universitat Heidelberg, 1995. Technical Report. Universitat Heidelberg.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.

Murray R. Spiegel. *Statistics*. McGraw-Hill, 1998.

Thomas Stützle and Marco Dorigo. A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans. Evolutionary Computation*, 6(4):358–365, 2002.

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