# DQL: a New Updating Strategy for Reinforcement Learning based on Q-learning

Carlos E. Mariano[1] and Eduardo F. Morales[2]

[1] Instituto Mexicano de Tecnología del Agua, Paseo Cuahunáhuac 8532, Jiutepec, Morelos, 62550, MEXICO
cmariano@tlaloc.imta.mx

[2] ITESM-Campus Cuernavaca, Paseo de la Reforma 182-A, Col. Lomas de Cuernavaca, Temixco, Morelos, 62589, MEXICO
emorales@campus.mor.itesm.mx

**Abstract.** In reinforcement learning an autonomous agent learns an optimal policy while interacting with the environment. In particular, in one-step Q-learning, with each action an agent updates its Q values considering immediate rewards. In this paper a new strategy for updating Q values is proposed. The strategy, implemented in an algorithm called DQL, uses a set of agents all searching the same goal in the same space to obtain the same optimal policy. Each agent leaves traces over a copy of the environment (copies of Q-values), while searching for a goal. These copies are used by the agents to decide which actions to take. Once all the agents reach a goal, the original Q-values of the best solution found by all the agents are updated using Watkins' Q-learning formula. DQL has some similarities with Gambardella's Ant-Q algorithm [4], however it does not require the definition of a domain dependent heuristic and consequently the tuning of additional parameters. DQL also does not update the original Q-values with zero reward while the agents are searching, as Ant-Q does. It is shown how DQL's guided exploration of several agents with selected exploitation (updating only the best solution) produces faster convergence times than Q-learning and Ant-Q on several testbed problems under similar conditions.

## 1 Introduction

Reinforcement learning is an on-line technique that approximates dynamic programming. The external environment is modeled as a discrete-time, finite state, Markov decision process. Each action is associated with a reward. The task of reinforcement learning is to maximize the long-term discounted reward per action.

Reinforcement learning has been recently applied to multi agent settings. The main purpose is to coordinate agents to complete a task. In coordination problems, each agent is responsible for a portion of the problem, and most of the time, decisions of an agent affect other agents' performance or solution. Examples include the solution of network routing problems in [6] and coordination

games such as soccer [7]. Multi agent reinforcement learning have also been used in pursuit games, where a 'hunter' tries to capture a 'prey'. In these problems, agents share sensations of the location of the 'prey', communicate its location to its partners and update their relative location in order reach the 'prey' [13]. Price and Boutilier [10] proposed a method called *implicit imitation*. In this approach apprentice agents learn from the experience of mentor agents about its own capabilities in unvisited parts of the space. Imitation is performed extracting a model from the experienced agent behavior. This approach was proved in the solution of mazes using model based reinforcement learning algorithms, speeding learning dramatically. Other interesting problems, solved using multi agent reinforcement learning, are those known as *n-player cooperative repeated games*. In these problems agents interact in a limited resource environment selecting actions that maximize reward. The chosen actions constitute a joint action. Each joint action is associated with a reward function; the decision problem is cooperative since there is a single reward function reflecting the utility assessment of all the agents. Agents must cooperate in order to select those actions representing the maximal individual and team benefit. Some approaches to establish cooperative behavior between agents for these kind of problems include [1, 2, 5].

In most of these approaches, single agent reinforcement learning methods are applied without much modification. In this paper, we propose an alternative strategy for updating value functions. The main motivation behind this research is to improve the convergence times of Q-learning with a distributed reinforcement learning setting, where a set of "agents" have the same goal, and together "cooperate" by leaving traces to find an optimal policy for the same problem. The hypothesis is that using more exploration with a set of agents and a controlled exploitation, by leaving traces between agents and reinforcing only the best solution proposed by the agents, produces faster convergence times.

DQL performance was compared against Q-learning [14] and Gambardella and Dorigo's Ant-Q algorithm [4], which is a distributed reinforcement learning algorithm used in the solution of the traveling salesman problem. The three algorithms were tested on several problems over the whole range of the $\alpha$ and $\gamma$ parameters used in the Q-learning formula. It is shown that DQL has faster convergence times than one-step Q-learning and Ant-Q under similar conditions.

The paper is organized as follows. Section 2 gives a brief overview of Q-learning and Ant-Q. Section 3 describes DQL. Section 4, presents the four test problems used to measure the algorithms' performance and discusses the main results. Finally, Section 5 concludes and gives future research directions.

## 2 Q-Learning

In this study, each reinforcement learning agent uses the one-step *Q-learning* algorithm [14]. Its learned decision policy is determined by the state-action pair value function, $Q(s, a)$, which estimates long-term discounted rewards for each state-action pair. Given a current state $s \in \mathcal{S}$ and available actions $a_i \in \mathcal{A}_s$, a Q-learning agent selects most of the time an action $a$ with the highest estimated

$Q(s,a)$ and with a small probability $\varepsilon \approx 0$, selects an alternative action. The agent then executes the action, receives an immediate reward $r$, and moves to the next state $s'$.

In each step, the agent updates $Q(s,a)$ by recursively discounting future utilities and weighting them by a positive learning rate $\alpha$:

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

where $(0 \leq \gamma \leq 1)$ is a discount parameter.

As an agent explores the state space, its estimate $Q$ improves gradually, and, eventually, each $\max_{a' \in \mathcal{A}'_s} Q(s',a')$ approaches: $E\left\{\sum_{n=1}^{\infty} \gamma^{n-1} r_{t+n}\right\}$. Here $r_t$ is the reward received at time $t$ due the action chosen at time $t-1$. Watkins and Dayan [15] have shown that this Q-learning algorithm converges to an optimal decision policy for a finite Markov decision process.

### 2.1 Ant-Q

An interesting distributed reinforcement algorithm, originally proposed by Gambardella and Dorigo [4], is Ant-Q. Ant-Q was used to solve traveling salesman problems and can be seen as an improvement over a previous system called ant systems [3]. The general idea of Ant-Q is to use a set of agents searching for the same best policy. Following an analogy with ant colonies, each agent updates its Q values, as in Q-learning, but without considering any reward ($r = 0$ in Eq. 1), after executing each action. This updating represent traces that can be followed by other agents. Once all the agents reach a goal (an episode), state-action pair evaluation functions of the best solution are updated using a delayed reward ($r \neq 0$) as expressed in Eq. 1. This means that some Q values will be updated several times on each episode, first without rewards by all the agents that followed the same state-action pair, and once more with rewards if the state-action pair is part of the best path. This repeated updating is not clearly justified and is difficult to prove if the convergence properties of Q-learning still hold.

Ant-Q introduced several additional mechanisms to the Q-learning framework. In particular, the selection policy is defined as a combination of a domain dependent heuristic function ($HE(s,a)$) and the best Q-values. This combination introduces two new parameters ($\delta$ and $\beta$) that estimate the relevance of $HE(s,a)$ with respect to $Q(s,a)$ values and that need to be tuned for each particular application domain. In general an $\epsilon$-greedy strategy is used and $HE$ is combined with $Q$ values as follows: $argmax_a\{AQ(s,a)^{\delta} \times HE(s,a)^{\beta}\}$.

## 3 DQL

DQL follows similar ideas of Ant-Q but without loosing the main properties of Q-leaning nor introducing extra parameters or heuristics. The general ideas, and main differences with Ant-Q, are that it does not use any domain dependent

heuristic (and consequently no additional parameters) and it updates the Q-values only once (for the best solution found by all the agents).

DQL allows more exploration, as several agents are searching at the same time, and promotes better exploitation, since the updates on the Q-values are performed only over the best solutions[1].

All the agents have access to a temporary copy of the state-action pair evaluation functions ($Q_C(s,a)$). Each time an agent has to select an action, it looks at this copy and decides, based on its information, which action to take. Once the agent performs the selected action, it updates the copy of the state-action value pair using Eq. 2, where $Q_C(s,a)$ represents a copy of the original $Q(s,a)$ pairs.

$$Q_C(s,a) \leftarrow Q_C(s,a) + \alpha \left[ \gamma \max_{a' \in \mathcal{A}_s} Q_C(s',a') - Q_C(s,a) \right] \tag{2}$$

This is similar to what Ant-Q does, however in this case the updates are performed over copies of the original Q values and the original Q-values are consequently not affected at this stage. All the agents are moved one step at a time, updating and sharing their common $Q_C$ values until reaching a stopping criterion. The agents use the copies of the Q values to decide which actions to take following an $\epsilon-$*greedy* policy. When all the agents have found a solution the Q value copies are discarded and the state-action pairs considered in the best solution receive a reward which reinforce their values according to Eq. 1. This updates the original Q-values from which a new copy is created for the next cycle. The whole process is repeated until reaching a termination criterion (see Table 1).

**Table 1.** DQL algorithm.

Initialize $Q(s,a)$ arbitrarily
Repeat (for $n$ episodes)
  Initialize $s$, copy $Q(s,a)$ to $Q_C(s,a)$
  Repeat (for each step of episode)
    Repeat (for $m$ agents)
      Take action $a$, observe $r$, $s'$
      $Q_C(s,a) \leftarrow Q_C(s,a) + \alpha\,[\gamma \max_{a'} Q_C(s',a') - Q_C(s,a)]$
      $s \leftarrow s'$;
    Until $s$ is terminal
  Evaluate the $m$ proposed solutions
  Assign rewards to the best solution found and
    update the Q values:
  $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

[1] In the tested problems, the best solution of one episode is the shortest path found by one agent in that episode.

All the agents act on the same environment and have access to the same $Q$ and $Q_C$ values. The copies of the Q values are used as guidances to the agents of what seems to be promising states. However, only the best solution found by all the agents receives an actual reward. There are two main differences with respect to Ant-Q:

- Partial updates are performed over copies of the Q-values avoiding multiple updates with and without rewards.
- There is no need to define a domain dependent heuristic or to tune extra parameters as in Ant-Q.

The main motivation behind DQL is that it allows:

- More exploration as more agents are used during search
- More exploitation as only relevant (best) solutions are effectively rewarded

The hypothesis is that this alternative strategy for updating value functions achieves, in general, faster convergence times than one-step Q-learning, regardless the values of $\alpha$ and $\gamma$. To test this hypothesis, we performed several experiments over four problems with different complexity and nature, comparing Q-learning, Ant-Q and DQL performance. Although, the tests were performed of deterministic state transition domains, our framework can also be applied to stochastic state transition functions.

## 4 Experimental Results

All the experiments were performed on the same machine and the algorithms were similarly coded by the same author[2]. Although DQL and Ant-Q use multiple agents the algorithms are implemented sequentially.

Two maze problems were first considered as they are problems where Q-learning normally shows good performance. For these problems the algorithms were tested over all possible $\alpha$ and $\gamma$ values with 0.25 increments. $\varepsilon = 0.1$ was considered for the three algorithms in both maze problems. Each experiment was performed thirty times and we report the mean CPU time, mean number of episodes, and mean number of steps per episode[3]. Algorithm execution stops when the optimal policy (solid lines in Figure 1) is reached in five consecutive episodes.

As mentioned earlier, the Ant-Q algorithm was designed for the solution of traveling salesman problems (TSPs). Two TSP instances previously solved with Ant-Q are also included in the tests. The same parameter values and stopping criteria used with Ant-Q were used for DQL and Q-learning. Tables of results include best solution found, standard deviation of solutions, the mean of all the solutions, and the mean CPU time to reach the stopping criterion. In this case, every algorithm was executed 15 times over 200 episodes.

[2] All algorithms are coded in C++.

[3] The mean number refers to the mean of all the solutions found at a particular episode.

### 4.1 Grid world with wind

The first experiment was run on the windy grid world shown in Figure 1 left. The objective is to find the optimal path from $S$ to $G$ considering a wind force, which shifts upwards the resulting state when moving horizontally, the strength of which varies from column to column as shown at the bottom of Figure 1. For instance, moving horizontally (either left or right) from a square which has a "wind force" of 1 (indicated at the bottom of Figure 1), causes the agent to move one square above its intended destination. However, moving vertically (either down or up) does not produce any effect[4]. Ant-Q and DQL were both run with 3 agents. Ant-Q was tested with and without a heuristic. The results are shown in Figures 2 and 3 without heuristic for Ant-Q.

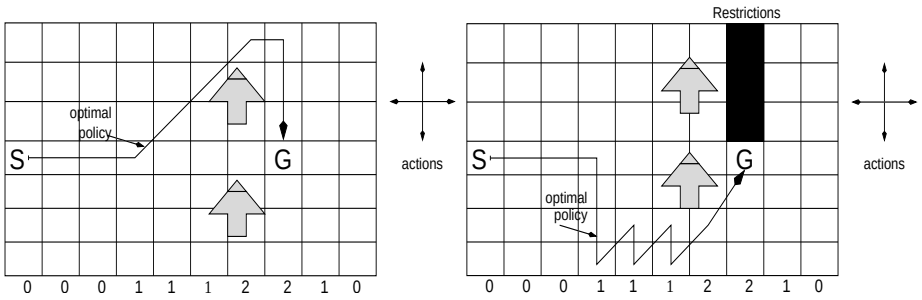


**Fig. 1.** Grid world in which horizontal movement is altered by a location-dependent upward "wind" (left) and windy world with restrictions (right).

Results from the three algorithms are plotted using three different line types, dashed for Ant-Q, dash-dot for Q-learning, and continuous for DQL. There are four lines for each algorithm, one for each value of $\gamma$. The + symbol correspond to $\gamma = 0.25$, $\circ$ to $\gamma = 0.5$, $\diamond$ to $\gamma = 0.75$, and $\Box$ to $\gamma = 1.0$.

Figure 2 shows the mean CPU time required for each algorithm to reach the stopping criterion. For Ant-Q and DQL this time corresponds to the mean time required for all the solutions found at each episode. As can be seen from the results, both Ant-Q and DQL clearly outperformed Q-learning for all the tested values of $\alpha$ and $\gamma$, significantly reducing the convergence times.

Figure 3 shows the mean number of episodes and the mean number of steps per episode required for the three algorithms to reach the stopping criterion. For these two metrics DQL performance was the best of the three algorithms, and Q-leaning was able to outperform Ant-Q for $\alpha \geq 0.5$.

The previously described results are for Ant-Q without using any heuristic. When Ant-Q was tested using as heuristic the inverse of the Manhattan distance,

[4] An agent is not allowed to move outside the borders.

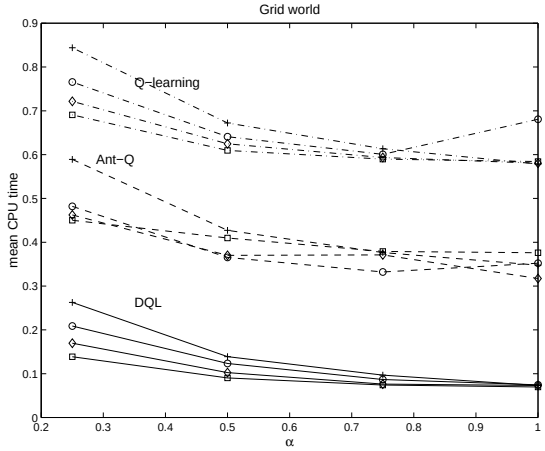


**Fig. 2.** Mean CPU time in seconds to reach the optimal solution five consecutive episodes.

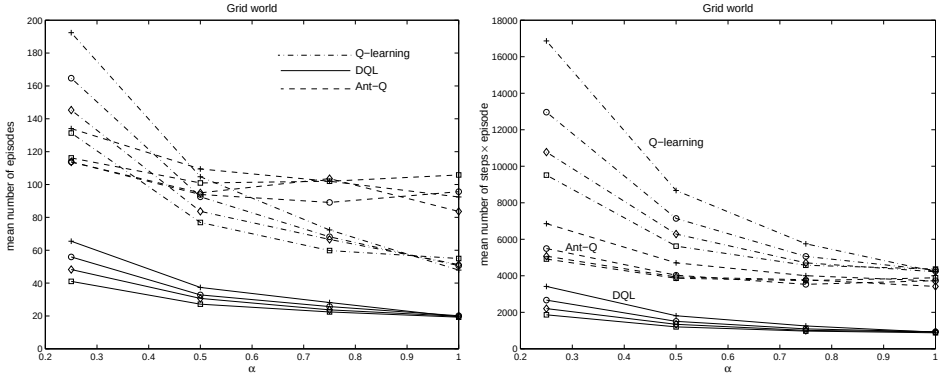


**Fig. 3.** Mean episodes (left) and steps per episode (right) required to reach the optimal solution.

it was not able to converge[5] with three different combinations of $(\delta, \beta)$: $(1,1)$ Q-values and heuristic function equally important, $(2,1)$ Q-values more important than the heuristic function, and $(1,2)$ heuristic function more important than Q-values. Although, the heuristic used may be reasonable for some maze problems, it is clear that in general, finding a suitable heuristic may be a very difficult task.

We also decided to test a variant of DQL, called DQL-2, where each agent performs a complete episode before starting with the next agent. That is, performing $m$ episodes in sequence without sharing information while performing the task. Figure 4 compares the mean CPU times of DQL (here as DQL-1) against this "episodic" updating approach (DQL-2). As it can be appreciated in the figure, sharing information while performing a task reduces convergence times. Although not shown in the paper, due to restrictions in space, similar results were observed in the other problems.

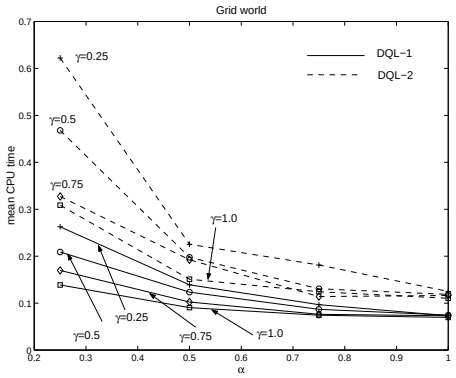


**Fig. 4.** Mean CPU times in seconds between DQL and an "episodic" variant (DQL-2)

We also measured the average number of total updates of Q values ($Q_C + Q$) in DQL against the number of Q updates of Q-learning (see Figure 5). As can be seen in the figure, although DQL updates a larger number of total Q-values, it converges faster. We believe that the extra information shared by the agents during the process helps to reduce convergence times. Similar behavior was observed on the other problems.

### 4.2 Grid world with wind and trap

This problem was designed to generate a more difficult maze. An obstacle blocking the optimal path is included to the windy grid world, forcing agents to search for an alternative route. Figure 1 right shows the maze and the optimal policy

[5] Reach the optimal policy five consecutive episodes before reaching 500,000 transitions.

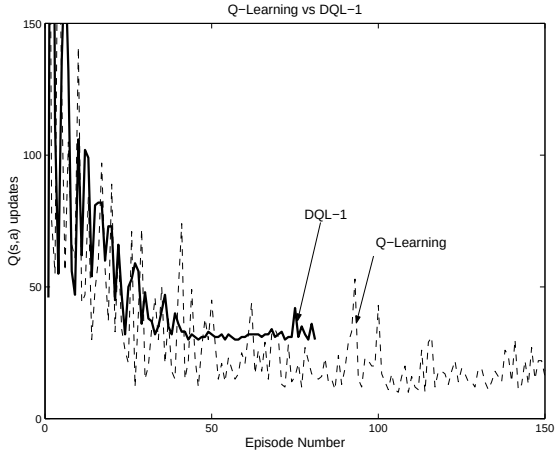


**Fig. 5.** Average number of total Q-value updates for DQL and Q-learning.

that agents must find. The same operation conditions and parameters used in the previous maze were considered.

Figures 6 and 7 show the measures for the three metrics obtained with the three algorithms under study. Again, the figures show only the performance of Ant-Q without heuristic as it was not able to converge with the Manhattan distance heuristic. In Figure 6 it can be observed that Ant-Q is able to outperformed DQL mean CPU time for some combinations of $\alpha$ and $\gamma$: ($\alpha = 0.25, \gamma \neq 0.25$), $\alpha = 0.5, \gamma = 0.5$, and ($\alpha = 1, \gamma = 0.5, 1.0$). It shows, however, to be much more dependent on the values of these parameters. On the other hand, DQL shows a more stable behavior in relation to the number of episodes and steps per episode required for the agents to satisfy the stopping criterion.

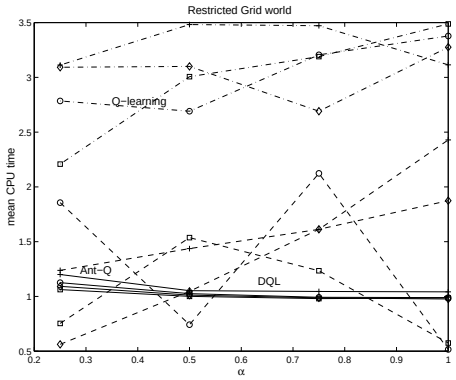


**Fig. 6.** Mean CPU time in seconds to reach the optimal solution five consecutive episodes.

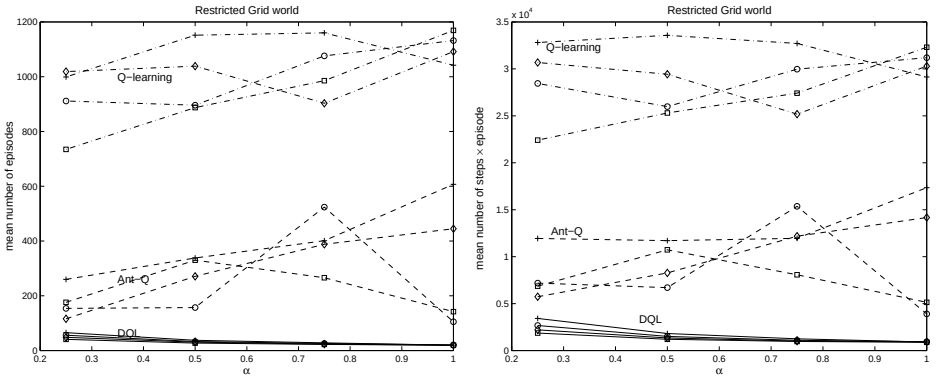


**Fig. 7.** Mean episodes (left) and steps per episode (right) required to reach the optimal solution.

### 4.3 Traveling Salesman Problems

Ant-Q was originally developed to solve instances of TSP. For them, the authors of Ant-Q reported results where Ant-Q outperformed several alternative algorithms. We took two instances of TSP with the same settings used in the original Ant-Q paper [4]. The first problem is the 30 cities symmetric TSP known as *Oliver30* proposed in [9], and the second problem is the 48 cities asymmetric TSP known as *Ry48p* proposed in [11].

Ant-Q parameters for the pseudo random proportional action choice rule were the same used by Gambardella, that is, $\beta = 2.0$, $\delta = 1.0$, $\alpha = 0.1$, $\gamma = 0.3$, and $HE(i,j) = 1/d_{i,j}$, being $i, j$ cities and $d_{i,j}$ the distance between them. For DQL and Q-learning, the same values for the $\alpha$ and $\gamma$ parameters were used. $Q(s,a)$ values were initialized to the inverse of the number of cities times the average length of edges, and an $\varepsilon$-greedy selection policy, with $\varepsilon = 0.1$ [6]. The algorithms considered 200 steps or transitions for *Oliver30* problem, and 600 for *ryp48*. The number of agents were the same in DQL and Ant-Q, 30 for *Oliver30* and 48 for *Ry48p*. For Q-learning a single agent was used.

The performance was evaluated repeating each trial 15 times. We report the average performances. The CPU time correspond to the average running times to reach the best result.

In the TSP problems all strategies found the same best solution. Although Ant-Q was specially "tuned" for this type of problems it did not show the best performance. It is also interesting to note that Q-learning show the lowest average solution for Ry48p. It is however clear from the results, that the average convergence times of DQL are much smaller than the other strategies. In these particular cases, the heuristic function added to Ant-Q was useful for reducing

[6] This policy is equivalent to consider $q_0 = 0.9$ in the original Ant-Q algorithm

**Table 2.** Results for the two TSPs.

| | | *Oliver30* | *Ry48p* |
|---|---|---|---|
| **Best** | | 423.74 | 14422 |
| **DQL** | Mean | 424.61 | 14600 |
| | Std. Dev. | 3.25 | 100 |
| | CPU | 3.48 | 13.29 |
| **Q-learning** | Mean | 425.32 | 14520 |
| | Std. Dev. | 4.12 | 150 |
| | CPU | 25.92 | 97.32 |
| **Ant-Q** | Mean | 424.92 | 14750 |
| **HE** | Std. Dev. | 3.7 | 189 |
| | CPU | 6.39 | 32.39 |
| **Ant-Q** | Mean | 435.43 | 15365 |
| **no HE** | Std. Dev. | 12.7 | 210 |
| | CPU | 46.84 | 76.98 |

convergence times and standard deviations (which was not the case for the grid world problems).

## 5 Conclusions and Future Work

This paper introduces a new strategy for updating Q values implemented in an algorithm called DQL. DQL uses a set of agents searching the same goal in the same space. Traces (copies of Q-values updated without rewards) are used to guide the exploration of agents. The original Q values of only the best solution found by all the agents is updated using the one-step Q-learning formula. It was shown how DQL's guided exploration of several agents with selected exploitation (updating only the best solution) produces faster convergence times than Q-learning and Ant-Q on several testbed problems under similar conditions.

The heuristic and extra parameters needed by Ant-Q does not seem to be producing any benefits. Additionally, selecting a good heuristic can be a difficult task. DQL, on the other hand, does not require extra parameters and shows, in general, better convergence times.

DQL updating strategy is performed only on the best path among $m$ solutions. In order to preserve the convergence properties of Q-learning, we need to show that all the Q-value state-action pairs have a non zero probability of being updated. This is part of our future work.

Before trying a parallel version, which seems a natural extension, we would like to perform more tests and compare the results against different strategies for updating Q values, such as Monte Carlo. It will also be interesting to run DQL without updating the copies of the Q-values to assess its influence in the results (which is like running Q-learning several times without updating and then update the best results found so far).

## Acknowledgments

Thanks to the anonymous reviewers for their helpful comments on the initial draft of this paper. The first author was supported by IMTA. This research was supported by CONACyT under grant 33000-A.

## References

1. Boutilier, C.: Sequential Optimality and Coordination in Multi agent Systems, In *Proc. of IJCAI-99*, Stockholm, Sweden, 1999.
2. Claus, C., Boutilier, C.: The Dynamics of Reinforcement Learning in Cooperative Multiagents Systems, In *Proc. of AAAI-97 Multiagent Learning Workshop*, pg. 13-18, Providence, 1997.
3. Dorigo, M.: *Optimization, Learning, and Natural Algorithms*, PhD thesis, Politecnico da Milano, Italy, 1992.
4. Gambardella, L., M., Dorigo, M.: Ant-Q: A reinforcement Learning Approach to the Traveling Salesman Problem, In *Proceedings of the 12th International Conference on Machine Learning*, pp. 252-260, Morgan Kaufmann, 1995.
5. Hu, J., Wellman, M.: Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm, In *Proc. 15th Int. Conf. on Machine Learning*, pp. 242-250, Morgan Kaufmann, 1998.
6. Littman, M., Boyan, J.: A Distributed Reinforcement Learning Scheme for Network Routing, In *Proc. Int. Workshop on Applications of Neural Networks to Telecommunications*, pp. 45-51, J. Alspector, et al., (eds.), Lawrence Erlbaum, Hillsdale, NJ, 1993.
7. Littman, M.: Markov Games as a Framework for Multiagent Reinforcement Learning, In *Proc. 11th Int. Conf. on Machine Learning*, pp. 157-163, New Brunswick, NJ, 1994, Morgan Kaufmann.
8. Mariano, C., Morales, E.: A New Distributed Reinforcement Learning Algorithm for the solution of Multiple Objective Optimization Problems, In O. Cairo et al., eds. *Lecture Notes in Artificial Intelligence*, 1793:212-223, April 2000.
9. Oliver, I., Smith, D., Holland, J.R.: A study of Permutation Crossover Operators on the Traveling Salesman Problem, In *Proc. 2nd Int. Conf. an Genetic Algorithms*, pp. 224-230, J.J. Grefenstette (ed.), Lawrence Erlbaum, Hillsdale, NJ, 1987.
10. Price, B., Boutilier, C.: Implicit Imitation in Multiagent Reinforcement Learning, In *Proc. 16th Int. Conf. on Machine Learning*, pp. , 1999.
11. Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications, Springer Verlag, Berlin, 1994.
12. Sutton, R., Barto, A.: *Reinforcement Learning an Introduction*, MIT Press, Cambridge, MA, 1998.
13. Tan, M.: Multiagent Reinforcement Learning: Independent vs. Cooperative Agents, In *Proc. 10th Int. Conf. on Machine Learning*, pp. 330-337, Amherst, MA, 1993.
14. Watkins, C.: *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, MA, 1978.
15. Watkins, C., Dayan, P.: Q-Learning, *Machine Learning*, 3:279-292, 1992.