Journal of Intelligent and Robotic Systems (2005) 43: 161–174
DOI: 10.1007/s10846-005-5137-x

A Reinforcement Learning Algorithm in Cooperative Multi-Robot Domains

FERNANDO FERNÁNDEZ[⋆] and DANIEL BORRAJO[⋆⋆]
Universidad Carlos III de Madrid, Avda/de la Universidad 30, 28911-Leganés, Madrid, Spain; e-mail: ffernand@inf.uc3m.es, dborrajo@ia.uc3m.es

LYNNE E. PARKER
University of Tennessee, 203 Claxton Complex, 1122 Volunteer Blvd, Knoxville, TN 37996-3450, U.S.A.; e-mail: parker@cs.utk.edu

(Received: 2 April 2004; in final form: 16 March 2005)

Abstract. Reinforcement learning has been widely applied to solve a diverse set of learning tasks, from board games to robot behaviours. In some of them, results have been very successful, but some tasks present several characteristics that make the application of reinforcement learning harder to define. One of these areas is multi-robot learning, which has two important problems. The first is credit assignment, or how to define the reinforcement signal to each robot belonging to a cooperative team depending on the results achieved by the whole team. The second one is working with large domains, where the amount of data can be large and different in each moment of a learning step. This paper studies both issues in a multi-robot environment, showing that introducing domain knowledge and machine learning algorithms can be combined to achieve successful cooperative behaviours.

Key words: reinforcement learning, function approximation, state space discretizations, collaborative multi-robot domains.

1. Introduction

Reinforcement Learning (Kaelbling et al., 1996) allows one to solve very different kinds of tasks by representing them as trial and error processes where single reinforcement signals indicate the goodness of performing actions at states. There are many different tasks that can be represented in this way, and they range from board games such as backgammon (Tesauro, 1992), to robot behaviours (Mahadevan and Connell, 1992). In these cases, the designer only needs to define the set of possible states, the set of possible actions, and typically a delayed reinforcement signal so that any reinforcement learning algorithm, such as *Q-Learning* (Watkins, 1989), can be applied.

[⋆] This work has been partially funded by a grant from Spanish Science and Technology Department.
[⋆⋆] This work has been partially funded by grants from Spanish Science and Technology Department number TAP1999-0535-C02-02, and TIC2002-04146-C05-05.

Multi-robot learning (Stone and Veloso, 2000; Balch and Parker, 2002) is another area where the application of reinforcement learning could produce improvements if it could be studied from the reinforcement learning perspective. However, to apply reinforcement learning to such domains is not easy, especially when they are *inherently cooperative* (Parker, 2002) (i.e., where the utility of the action of one robot is dependent upon the current actions of the other team members), because credit assignment is hard to define. Another problem is that the definition of the state of a team member can be very different, given that it can introduce local information of such a member, but it could even introduce information communicated from other team members. So, this information could be incomplete in different moments of the learning task because of its sensing capabilities, resulting in uncertainty about the whole state. In other words, it could transform a Markov Decision Process to a Partially Observable Markov Decision Process (Puterman, 1994). Furthermore, given that a lot of information can be used, the state space of each robot grows, requiring generalization techniques (Santamaría et al., 1998), that allow the generalization of knowledge acquired from a limited experience to any situation in the whole state space. Several other problems can be added to the previous ones, such as non-determinism in actions, limited training experience, etc.

In (Fernández and Parker, 2001), it was shown that a cooperative task can be mapped to a single reinforcement learning problem, and that the behaviour obtained from local information can be improved by increasing the perception of the robots to include information of other robots and refining the reinforcement signal to take into account this new information. The task used in that experimentation was the Cooperative Multi-robot Observation of Multiple Moving Targets task (CMOMMT) (Parker, 2002), that was redefined as a reinforcement learning domain, and the reinforcement learning algorithm applied was the VQQL algorithm (Fernández and Borrajo, 2000). This paper explores the application of a new reinforcement learning technique, the ENNC-QL algorithm (Fernández and Borrajo, 2002), in such a domain, comparing the results achieved with previous approaches. This algorithm is a mixed model between generalization methods based on supervised function approximation (Bertsekas and Tsitsiklis, 1996) and generalization methods based on state space discretization (Moore and Atkeson, 1995). Experiments in robot navigation domains illustrate that the mixed model is able to obtain better results than the two components separately.

Thus, the goal of this paper is two-fold. First, to verify whether the ENNC-QL algorithm is able to scale-up from learning behaviours in single robot domains (such as the robot navigation task presented in (Fernández and Borrajo, 2002)) to learning cooperative behaviours in multi-robot domains (such as the CMOMMT domain). Second, to verify whether the adaptation of the CMOMMT domain as a reinforcement learning domain presented in (Fernández and Parker, 2001) is general enough to be solved with different reinforcement learning techniques, in this case, ENNC-QL.

The next section introduces the ENNC-QL algorithm, while Section 3 describes the CMOMMT domain. Section 3 also briefly describes some previous approaches to this domain, introducing the view of the domain as a reinforcement learning domain. Section 4 shows the experiments performed with the ENNC-QL algorithm on the CMOMMT domain, comparing the results with the previously described approaches. Finally, Section 5 presents the main conclusions and future research.

2. ENNC-QL

This section describes the ENNC-QL algorithm, which can be defined as a reinforcement learning method based on discretizing the state space environment, but reducing the effect of losing the Markov property, and hence, reducing the introduction of non-determinism (Fernández and Borrajo, 2002). This method is closely related to other methods based on the supervised approximation of the value functions, so it can be considered as a hybrid model. The algorithm is based on an iterative process that allows the computation of both the discretization and the action-value function at the same time. In each iteration, new regions are computed from the value function approximation borders, computed in the previous iteration, and the value function approximation is also recomputed from the optimal local discretization computed at that moment.

The algorithm used for the supervised learning of the action-value function is the ENNC algorithm, briefly described next.

2.1. EVOLUTIONARY DESIGN OF NEAREST NEIGHBOUR CLASSIFIERS

1-Nearest Neighbour Classifiers (1-NN) are a particular case of k-NN classifiers that assign to each new unlabeled example e the label of the nearest prototype c from a set of N different prototypes previously classified (Duda and Hart, 1973). The main generic parameters of this sort of classifiers are: the number of prototypes to use, their initial position, and a smoothing parameter. The ENNC algorithm (Fernández and Isasi, 2002, 2004) is a 1-nearest neighbour classifier whose main characteristic is that it has a small number of user defined parameters: it only needs the number of iterations to run (say N). This means that none of the above parameters has to be supplied.

The algorithm starts with only one prototype in the initial set of prototypes. This set is modified iteratively by the execution of several operators in each execution cycle. At the end of the evolution, the classifier is composed of a reduced set of prototypes, which have been correctly labeled with a class value. The main steps of the algorithm are summarized as follows:

– Initialization. The initial number of prototypes is one. The method is able to generate new prototypes stabilizing in the most appropriate number in terms of a defined "quality" measure.
– Execute N cycles. In each cycle, execute the following operators:

Information Gathering. At the beginning of each cycle, the algorithm computes the information required to execute the operators. This information relates to the prototypes, their quality and their relationship with existing classes.

Labeling. Label each prototype with the most popular class in its region.

Reproduction. Introduce new prototypes in the classifier. The insertion of new prototypes is a decision that is taken by each prototype; each prototype has the opportunity of introducing a new prototype in order to increase its own quality.

Fight. Provide each prototype the capability of getting training patterns from other regions.

Move. Realloce of each prototype in the best expected place. This best place is the centroid of all the training patterns that it represents.

Die. Eliminate prototypes. The probability to die is inversely proportional to the quality of each prototype.

Once this classification approach is presented, the next section shows how to use it in a reinforcement learning method.

2.2. THE ENNC-QL ALGORITHM

We define the learning problem in ENNC-QL as follows. Given a domain with a continuous state space, where an agent can execute a set of L actions $A = \{a_1, \ldots, a_L\}$, the goal is to obtain an approximation of the action value function $Q(s, a)$ (Bellman, 1957). Specifically, L approximations $\widehat{Q}_{a_i}(s)$, for $i = 1, \ldots, L$, are computed, given the action parameter a is extracted from the function Q.

A high level description of the algorithm is shown in Figure 1. The algorithm starts with an initialization step, where the $L\widehat{Q}_{a_i}(s)$ approximators are initialized, typically to 0. Given that the function approximator used, the ENNC algorithm, follows a nearest prototype approach, it can be considered that the prototypes of ENNC generate a state space discretization. So, the way of initializing L nearest prototype approximators to the 0 value is to create L nearest prototype classifiers of only one prototype labeled as 0.

The second step is an exploratory phase. This phase generates a set T of experience tuples, of the type $\langle s, a_i, s', r\rangle$, where s is any state, a_i is the action executed from that state, s' is the state achieved and r is the immediate reward received from the environment. This initial exploration is not the focus of this work, but different approaches could be used, from random exploration to human directed exploration (Smart, 2002).

From this initial set of tuples, an iterative process is executed, where the approximators $\widehat{Q}_{a_i}(s)$ are learned. Given that these approximators generate a set of prototypes, these prototypes can be considered to discretize the state space. Thus,

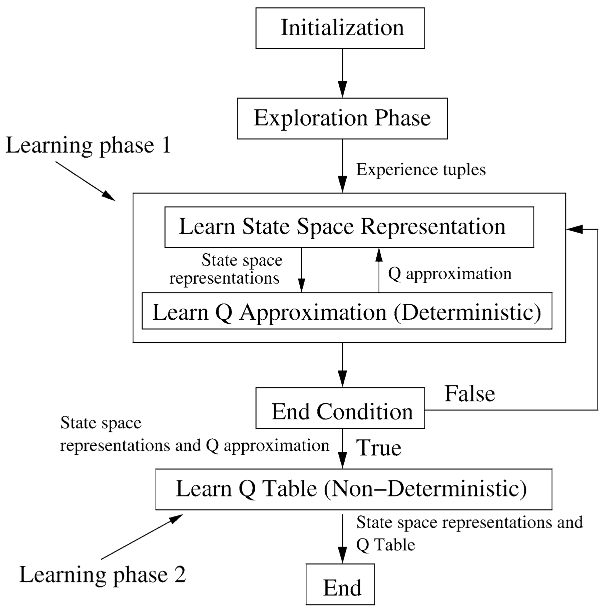
Initialization
Exploration Phase
Learning phase 1
Experience tuples
Learn State Space Representation
State space representations
Q approximation
Learn Q Approximation (Deterministic)
End Condition
False
True
State space representations and Q approximation
Learn Q Table (Non−Deterministic)
State space representations and Q Table
Learning phase 2
End

Figure 1. High level description of the ENNC-QL algorithm.

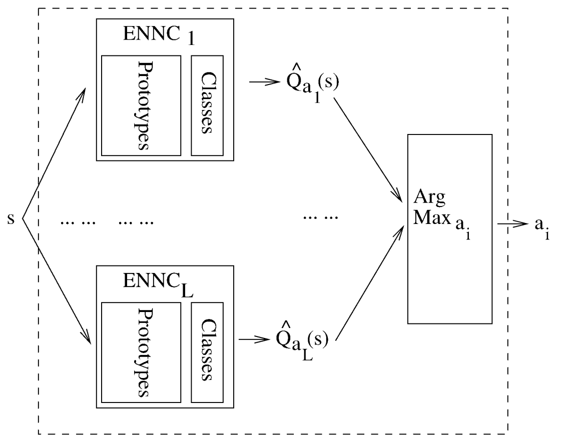

Figure 2. Function approximation view of the ENNC-QL algorithm in the first learning phase.

at the same time that the $\widehat{Q}_{a_i}(s)$ are computed, L state space discretizations, $S_{a_i}(s)$, are computed too, given that each $S_{a_i}(s)$ is composed of the prototypes of $\widehat{Q}_{a_i}(s)$ without the class labels. So, at the end of this iterative phase, the L new state space representations, as well as the approximation of the Q function, are obtained. The architecture of ENNC-QL in this first learning phase is shown in Figure 2.

In each iteration, from the initial set of tuples, and using the approximators $\widehat{Q}_{a_i}(s)$, $i = 1, \ldots, L$, generated in the previous iteration, the Q-Learning update rule for deterministic domains can be used to obtain L training sets, T_i^0, $i = 1, \ldots, L$, with entries of the kind $\langle s, q_{s,a_i}\rangle$ where q_{s,a_i} is the resulting value of applying the Q-Learning update function (Watkins, 1989) to each training tuple, i.e. $q_{s,a_i} = r + \gamma \max_{a_j \in A} \widehat{Q}_{a_j}(s')$. In this first iteration, $\widehat{Q}_{a_i}(s) = 0, i = 1, \ldots, L$, for all s, so the possible values for q_{s,a_i} depend only on the possible values of r. If we suppose the r values are always 0, except when a goal state is achieved, where a maximum reward of $r_{\max}$ is obtained, the only two values for q_{s,a_i} are 0 and $r_{\max}$ in the first iteration. However, in the following iteration, there will be experience tuples $\langle s, a_i, s' \rangle$, for which some $\widehat{Q}_{a'_i}(s')$ will be $r_{\max}$, so examples of the kind $\langle s, \gamma^1 r_{\max} \rangle$ will appear, and a new approximator will be learned with this new data. Repeating this process iteratively the whole domain will be learned from examples of the kind $\langle s, \gamma^t r_{\max} \rangle$, for $t = 0, \ldots, k$.

At the end of this phase, the approximation of the action-value function and the new state space representation have been computed. These new representations have a very important property. If we assume the original domain is deterministic, and that the ENNC classifier is able to exactly differentiate all the classes (the different values of the Q function), the new state space discretization satisfies the Markov property, and it does not introduce non-determinism, so it will accurately approximate the Q function.

However, the original state space representation may be stochastic, so the classifier may not be perfect, and errors could be introduced in the Q function approximation and the new state space representation. These facts motivate the second learning phase of the algorithm, that uses the state space discretizations obtained to learn a tabular representation of the Q function as Figure 3 shows.

The second learning phase helps to reduce the errors generated in the first phase because it uses the stochastic version of the Q-Learning update function, defined in Equation (1):

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]. \tag{1}$$

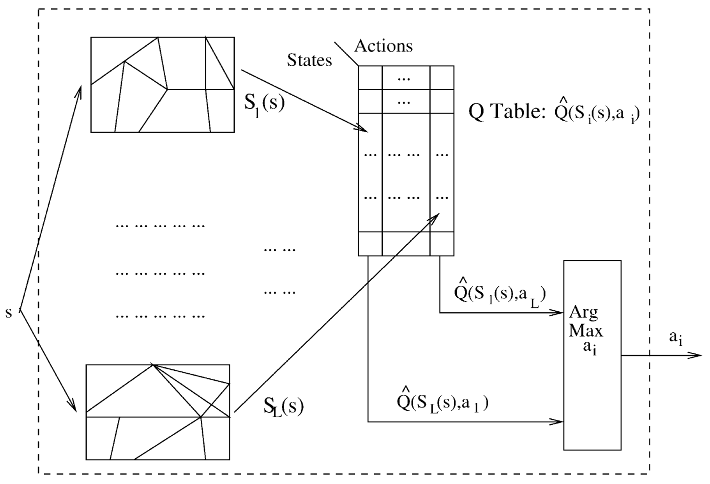

Figure 3. Architecture of ENNC-QL in the second learning phase.

3. Cooperative Multi-robot Observation of Multiple Moving Targets

The application domain used as a multi-robot learning test-bed in this research is the problem entitled *Cooperative Multi-robot Observation of Multiple Moving Targets* (CMOMMT), that is defined as follows (Parker, 2002). Given:

– $\mathcal{S}$: a two-dimensional, bounded, enclosed spatial region;
– $\mathcal{V}$: a team of m robot vehicles, $v_i, i = 1, 2, \ldots, m$, with 360° field of view observation sensors that are noisy and of limited range;
– $\mathcal{O}(t)$: a set of n targets, $o_j(t)$, $j = 1, 2, \ldots, n$, such that target $o_j(t)$ is located within region $\mathcal{S}$ at time t.

A robot v_i is *observing* a target when the target is within v_i's sensing range. Define an $m \times n$ matrix $B(t)$, as follows:

$$B(t) = [b_{ij}(t)]_{m\times n} \quad \text{such that}$$
$$b_{ij}(t) = \begin{cases} 1 & \text{if robot } v_i \text{ is } \textit{observing} \text{ target } o_j(t) \text{ in } \mathcal{S} \text{ at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

The goal is to develop an algorithm that maximizes the following metric A:

$$A = \sum_{t=1}^{T} \sum_{j=1}^{n} \frac{g(B(t), j)}{T},$$

where

$$g(B(t), j) = \begin{cases} 1 & \text{if there exists an } i \text{ such that } b_{ij}(t) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

That is, the goal of the robots is to maximize the average number of targets in $\mathcal{S}$ that are being observed by at least one robot throughout the mission that is of length T time units. Additionally, *sensor_coverage*(v_i) is defined as the region visible to robot v_i's observation sensors, for $v_i \in \mathcal{V}$. In general, the maximum region covered by the observation sensors of the robot team is much less than the total region to be observed. That is, $\bigcup_{v_i \in \mathcal{V}} \textit{sensor\_coverage}(v_i) \ll \mathcal{S}$. This implies that fixed robot sensing locations or sensing paths will not be adequate in general, and instead, the robots must move dynamically as targets appear in order to maintain their target observations and to maximize the coverage. Additionally, we do not assume that the number of targets is constant or known to the robots.

In (Parker and Touzet, 2000), some results are reported in the CMOMMT application. In that work, two main approaches were presented: a hand-generated solution and a learning approach. The hand-generated solution (called A-CMOMMT) was developed by a human engineer, and is based on weighted vectors of attraction from each robot to the targets and repulsion from other robots. The result of that approach is that each robot is attracted to nearby targets and is repulsed by nearby robots, with the movement of the robot calculated as the weighted summation of attractive and repulsive force vectors.

The learning approach presented in (Parker and Touzet, 2000) was called Pessimistic Lazy Q-Learning. It is based on a combination of lazy learning (Aha, 1997), Q-Learning, and a pessimistic algorithm for evaluating global rewards. This instance-based algorithm stores a set of situations in a memory in order to use them when needed. A pessimistic utility metric is used to choose the right action from this set of situations.

In (Fernández and Parker, 2001), the use of the VQQL model over this domain can be found. This algorithm is based on the unsupervised discretization of the state space, so tabular representations of the $Q(s, a)$ function can be applied (Fernández and Borrajo, 2000). Furthermore, that work defined the CMOMMT application as a delayed reinforcement learning problem as follows:

- In the CMOMMT domain, relevant input data consists of the locations of the targets and the other robots. However, at each moment, we likely have a partially observable environment, since not all targets and robots will be generally known to each robot. As an approximation, the approach can maintain information about the nearest targets and the nearest robots, using a mask when information is not known. So, the size of the input data depends on the number of targets and robots used as local information, and may differ in different experiments.
- Actions are discretized into eight skills, following the cardinal points: go North, go North–East, go East, etc. Additional actions can be introduced if desired. Then, if the agent is in a discretized state $\hat{s}$, and performs the action "go North", it will be moving until it arrives to a discretized state $\hat{s}' \neq \hat{s}$.
- The reinforcement function changes in the experiments, depending on the input data that is received. In most cases, positive rewards are given when targets are observed, so a higher reward is obtained with higher numbers of targets in view. This positive reward is counteracted in some experiments when other robots are in view, which is the criterion used to define whether or not the robots are collaborating. Therefore, negative reinforcements may be received if other robots are in the same viewing range. Furthermore, a delayed reinforcement approach has been followed, so reinforcements are only received at the end of each trial.

4. Experiments and Results

The experiments are aimed at comparing the application of the ENNC-QL model to the CMOMMT domain, to that of the VQQL model, following the same experimentation setup in (Fernández and Parker, 2001), described in Section 3. The only difference with those experiments is that length of actions, i.e. the time the robot is executing each action, is fixed, given that in this case, there are not discretized regions defining the size of the actions. So two different experiments are performed. In the first one (called local VQQL and local ENNC-QL, respectively), the only input data used is information on the furthest target in view of the robot. Thus, each RL state is a two component tuple storing the x and y component of the distance vector from the robot to the furthest target in view. Figure 4 shows the x and y

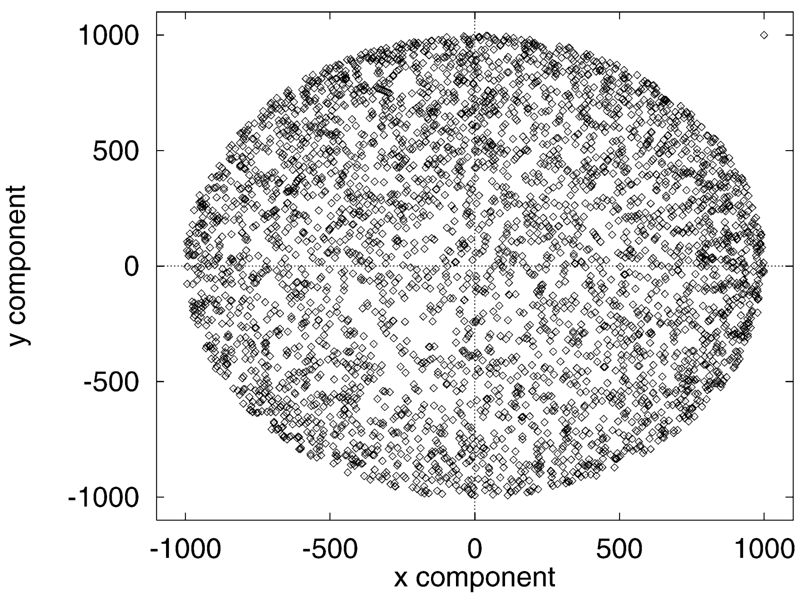

Figure 4. Distance vectors from the robot to the furthest target within viewing range.

components of such a state, for a set of states obtained from a random movement of a robot in the domain.

In this sense, we can see how this input data already introduces statistical information. For instance, the x component and y component are such that the distance vector is smaller than the range of view of the robots, except for the point (1000, 1000), which is the mask value used when the robot cannot see any target.

In this case, the delayed reinforcement function is the number of targets that the agent sees. The number of targets is 10, while the number of robots is one in the learning phase, and 10 in the test phase. In this case, no collaboration strategy has been defined in the experiment among the robots, and positive delayed reward is only given at the end of each trial. The reward is defined as the the number of targets under observation. Furthermore, if the robot loses all the targets, it receives a negative reinforcement, and remains motionless until some target enters its viewing range. The duration of each trial in the learning phase is 100 simulation steps.

The goal of the second experiment is to achieve a better performance by introducing collaborative behaviors among the robots. In this sense, given that the only signal that the robots receive in order to learn their behavior is the reinforcement signal, the collaboration must be implicitly introduced. To achieve this, the state space representation is increased in order to incorporate more information about targets and other robots. Thus, a state is composed of information about the nearest target within view, the furthest target within view, as well as the nearest robot. This increases the state vector from two to six components. Even if this number is not very high, it typically makes uniform discretization based reinforcement learning methods require an impractical amount of experience.

Adding this new information requires a change in the training phase as well, so the ten robots are present in the learning phase, even when only one of them is learning the behavior. In the test phase, all the robots will use the behavior learned by the first one.

Furthermore, in this approach (called collaborative VQQL and collaborative ENNC-QL, respectively) the reinforcement signal now incorporates a negative reward that is given at the end of each trial in order to achieve a collaborative behavior. This negative reward is based on whether or not the robot has another robot in its range of view. Thus, the reinforcement function for each robot i at the last moment of the trial, $r_i(T)$, is calculated in this way (following the notation introduced in Section 3):

$$r_i(T) = \left(\sum_{j=1}^{n} b_{ij}(T)\right) - k(T), \tag{2}$$

where

– $b_{ij}(t) = \begin{cases} 1 & \text{if robot } v_i \text{ is } \textit{observing} \text{ target } o_j(t) \text{ in } \mathcal{S} \text{ at time } t, \\ 0 & \text{otherwise,} \end{cases}$

– n is the number of targets,

– and $k(T)$ is a function whose value is 2 if the robot can see other robots, or 0 otherwise.

The basic idea is that the optimal behavior will be achieved when each robot follows a different target, to the greatest extent possible. This negative reward does not ensure this characteristic, but it approximates it. In future work, we would like to continue studying other reward functions and ways of modeling the domain.

All the approaches and the experiments performed are summarized in Figure 5, which shows the percentage of targets under observation for all the methods defined in a trial of 1000 units length. For the VQQL and the ENNC-QL algorithms, the value is averaged over 10 different trials, in order to avoid bias from initial situations.

First, the figure compares the results of the hand-generated solution and the Pessimistic Lazy Q-learning approach, along with two simple control cases, local (hand-made) solution and a random action selection policy (Parker, 2002). These results show that the pessimistic algorithm is better than the random and the local approaches, obtaining a 50% rate of performance. However, this result is far from the 71% achieved by the hand-generated solution (A-CMOMMT) of (Parker and Touzet, 2000).

For local VQQL, Figure 5 shows the results of the learning process for different state space sizes. In this case, with only 16 different states the robots achieve a 59% rate of performance; representations with a higher number of states do not provide higher performance improvements. When the collaborative strategy is used, a higher number of states is needed to achieve a good behavior because of the increment in the number of input attributes. For state space representations of only 64 states, around a 40% rate of performance is achieved. For 256 states,

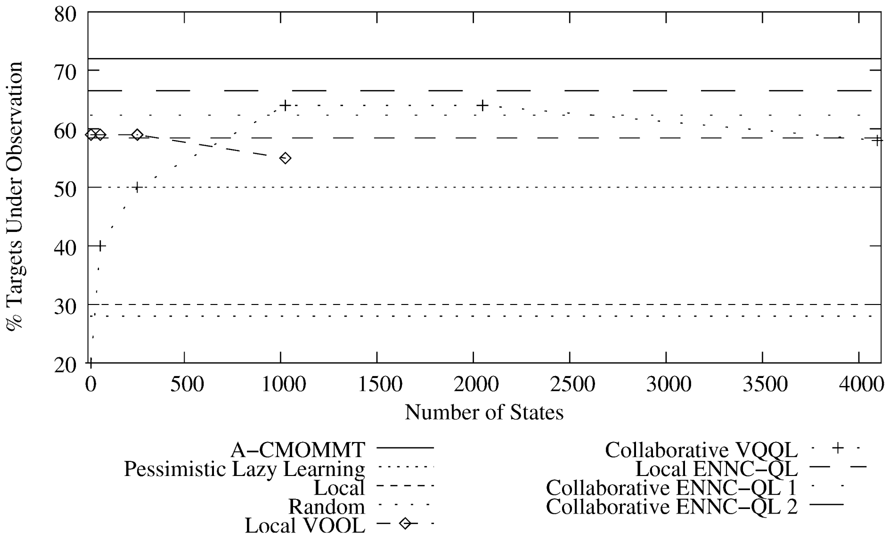

Figure 5. Results of different approaches on the CMOMMT application.

the performance is increased up to 50%, and for 1024 states, the 60% level of performance achieved previously is also obtained. The real improvement appears with 2048 states, where the percentage of targets under observation is 65% – five percentage points higher than in the previous experiment, and near the best hand-generated solution reported in (Parker and Touzet, 2000). Increasing the number of states does not improve performance, given that the problem of a very large number of states (and bad generalization) appears. Note also that the problem generally assumes many more targets than robots, so in general, we would not expect a performance of 100% of targets under observation.

VQQL is the only method of the ones described here that requires investigating different state space sizes. This is because if the different sizes are not tested, it is not possible to verify which will be the result, and depending on the problem, the optimal size may be different. The main advantage of ENNC-QL is that this value is automatically computed. That means that the designer does not need to worry about the complexity of the problem, and only needs to run the algorithm. Then, the algorithm outputs only one result, whose performance is the one shown in Figure 5.⋆

An interesting issue when using the ENNC-QL algorithm in this domain is that it only requires one iteration in its first learning phase to differentiate the areas with negative rewards from the areas with null or positive rewards because negative rewards are not propagated to the rest of the environment. In the second learning

⋆ However, given that the ENNC algorithm is stochastic, it may result in different values in different learning processes, so the results provided in Figure 5 are the average value of 5 different learning processes.

Execution	Prototypes	Success
1	107.2	54.12
2	91.37	61.42
3	110	58.22
4	90.5	58.88
5	110.62	59.62
Average	101.9	58.45
Std. Dev.	8.8	1.8

(a) Local ENNC-QL

Execution	Prototypes	Success (option 2)	Success (option 1)
1	187.37	67.41	66.44
2	192.87	66.96	61.34
3	192.12	65.99	60.5
4	196.5	66.38	62.35
5	149.75	65.91	61
Average	183.72	66.53	62.32
Std. Dev.	13.58	0.52	1.65

(b) Collaborative ENNC-QL

Figure 6. Results of different executions of the ENNC-QL algorithm.

phase, positive rewards will modify the approximation of the Q function obtained in the previous phase, refining the obtained policy.

The first result obtained is called Local ENNC-QL, where only the coordinates from the nearest target are used. The success achieved by this approach is 58.45% of targets under observation, for state space discretizations of around 100 states, so similar results to VQQL are achieved. However, when the data from the nearest robot and the furthest target are used too (solution called Collaborative ENNC-QL 1) the results increase up to the 62.33%, very close to VQQL. However, in this case, the number of states used is less than 200, instead of the more than 2,000 used with VQQL. So similar solutions are achieved, but with fewer states and automatically. In both cases, the behavior learned is the same for all the robots. However, if each robot is allowed to learn its own Q table in the second learning phase of the ENNC-QL algorithm (called Collaborative ENNC-QL 2), the performance increases to 66.53%, very close to the best hand made solution reported. Thus, ENNC-QL offers two main advantages over VQQL. On the one hand, the number of states generated is smaller. On the other side, the number of states is automatically computed, so parameter tuning is not required.

Figure 6 describes the results obtained in each of the five executions, showing that a small difference exists among them (represented by their standard deviations), but obtaining good results in all of them, taking into account both the success in solving the task and the average number of prototypes achieved for the state space discretizations.

5. Conclusions and Further Research

In this paper, we have shown how a cooperative multi-robot domain can be studied from a reinforcement learning point of view, only by defining a set of discretized actions, limiting the state space to use only attributes that the designer considers necessary, carefully defining a reinforcement function, and using a technique that allows generalization from limited experience to a continuous state space. Two main phases are required. In the first, the designer must choose the state space, taking into account information that s/he considers necessary. In the second, a method able to generalize must be used, because even if the designer chooses a reduced set of attributes defining a state, this value could still be very large. In this case, the ENNC-QL algorithm has been chosen, obtaining good results when compared with previous approaches, and showing that it can be successfully applied in continuous cooperative domains.

Future work has two main lines. The first is trying to find automatic methods for defining the right set of attributes to define the state, that, in the machine learning literature, is typically called feature selection (Tsitsiklis and Roy, 1996). The second research line is to define a correct reinforcement function from the set of features obtained in the previous step. In this sense, Inverse Reinforcement Learning (Ng and Russel, 2000) could help to learn the reinforcement function from the hand-generated solution, and then, try to learn an improved policy.

References

Aha, D.: 1997, *Lazy Learning*, Kluwer Academic Publishers, Dordrecht.

Balch, T. and Parker, L. E. (eds): 2002, *Robot Teams: from Diversity to Polymorphism*. A. K. Peters Publishers.

Bellman, R.: 1957, *Dynamic Programming*, Princeton Univ. Press, Princeton, NJ.

Bertsekas, D. P. and Tsitsiklis, J. N.: 1996, *Neuro-Dynamic Programming*, Athena Scientific, Bellmon, MA.

Duda, R. O. and Hart, P. E.: 1973, *Pattern Classification and Scene Analysis*, Wiley, New York.

Fernández, F. and Borrajo, D.: 2000, VQQL. Applying vector quantization to reinforcement learning, in: *RoboCup-99: Robot Soccer World Cup III*, Lecture Notes in Artificial Intelligence, Vol. 1856, Springer, Berlin, pp. 292–303.

Fernández, F. and Borrajo, D.: 2002, On determinism handling while learning reduced state space representations, in: *Proc. of the European Conf. on Artificial Intelligence (ECAI 2002)*, Lyon, France, July.

Fernández, F. and Isasi, P.: 2002, Automatic finding of good classifiers following a biologically inspired metaphor, *Computing Informatics* **21**(3), 205–220.

Fernández, F. and Isasi, P.: 2004, Evolutionary design of nearest prototype classifiers, *J. Heuristics* **10**(4), 431–454.

Fernández, F. and Parker, L.: 2001, Learning in large cooperative multi-robot domains, *Internat. J. Robotics Automat.* **16**(4), 217–226.

Kaelbling, L. P., Littman, M. L., and Moore, A. W.: 1996, Reinforcement learning: A survey, *J. Artificial Intelligence Res.* **4**, 237–285.

Mahadevan, S. and Connell, J.: 1992, Automatic programming of behaviour-based robots using reinforcement learning, *Artificial Intelligence* **55**(2/3), 311–365.

Moore, A. W. and Atkeson, C. G.: 1995, The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces, *Machine Learning* **21**(3), 199–233.

Ng, A. Y. and Russel, S.: 2000, Algorithms for inverse reinforcement learning, in: *Proc. of the Seventeenth Internat. Conf. on Machine Learning*.

Parker, L. and Touzet, C.: 2000, Multi-robot learning in a cooperative observation task, in: L. E. Parker, G. Bekey and J. Barhen (eds), *Distributed Autonomous Robotic Systems*, Vol. 4, Springer, Berlin, pp. 391–401.

Parker, L. E.: 2002, Distributed algorithms for multi-robot observation of multiple moving targets, *Autonom. Robots* **12**(3), 231–255.

Puterman, M. L.: 1994, *Markov Decision Processes – Discrete Stochastic Dynamic Programming*, Wiley, New York.

Santamaría, J. C., Sutton, R. S., and Ram, A.: 1998, Experiments with reinforcement learning in problems with continuous state and action spaces, *Adaptive Behavior* **6**(2), 163–218.

Smart, W. D.: 2002, *Making reinforcement learning work on real robots*, PhD Thesis, Department of Computer Science at Brown University, Providence, RI.

Stone, P. and Veloso, M.: 2000, Multiagent systems: A survey from a machine learning perspective, *Autonom. Robots* **8**(3).

Tesauro, G.: 1992, Practical issues in temporal difference learning, *Machine Learning* **8**, 257–277.

Tsitsiklis, J. N. and Van Roy, B.: 1996, Feature-based methods for large scale dynamic programming, *Machine Learning* **22**, 59–94.

Watkins C. J. C. H.: 1989, Learning from delayed rewards, PhD Thesis, King's College, Cambridge, UK.