LEARNING IN LARGE COOPERATIVE MULTI-ROBOT DOMAINS

F. Fernández*

L. E. Parker

Center for Engineering Science Advanced Research

Computer Science and Mathematics Division

Oak Ridge National Laboratory, Oak Ridge TN 37831-6355, USA

email: ffernand@scalab.uc3m.es, ParkerLe@ornl.gov

13th June 2001

Abstract

The development of mechanisms that enable robot teams to autonomously generate cooperative behaviours is one of the most interesting issues in distributed and autonomous robotic systems. In this paper, the application of reinforcement learning techniques to robot teams is studied, enabling the robot to learn cooperative behaviours based only on local information. The $VQQL$ technique is applied both to learn a state space representation of the environment and to learn a correct behaviour. The Cooperative Multi-robot Observation of Multiple Moving Targets (CMOMMT) application is used as a well-known domain in the robotics field, and results are compared with previous works, obtaining improved performance.

*Present Address: Universidad Carlos III de Madrid. Avda/ de la Universidad 30. Leganés. 28911, Madrid (Spain)

Keywords: Multi-robot systems, cooperative robotics, reinforcement learning, state space representation

1 Introduction

Researchers have recognized that an approach with high potential for simplifying the development of cooperative multi-robot teams is autonomous learning. Hence, much work is ongoing in the field of multi-agent learning (e.g., [27]). However, this previous research in multi-agent learning is not immediately applicable to multi-robot learning, because agent representations are usually symbolic, whereas robotic representations are sub-symbolic. Multi-agent learning is therefore somewhat simplified, because a model of the agent-world interaction can be provided by the operator. Developing similar models for multi-robot learning is very difficult.

While multi-robot learning is an interesting topic from a pure research perspective, from a practical engineering viewpoint, it only makes sense to incorporate learning into a system if it reduces the cost of system development or maintenance. Such a cost reduction can be realized if (1) the learning significantly reduces the complexity of multi-robot team programming, or (2) the learning allows the robot team to dynamically adapt its performance to a changing environment, rather than requiring total system reprogramming over time. Thus, our goal in addressing multi-robot learning is to both reduce the complexity of multi-robot team design and to allow the team to autonomously adapt to changes in its environment.

Particularly challenging domains for multi-robot learning are those tasks that are *inherently cooperative*. By this, we mean that the utility of the action of one robot is dependent upon the current actions of the other team members. Inherently cooperative tasks cannot be decomposed into independent subtasks to be solved by a distributed robot team. Instead, the success of the team

throughout its execution is measured by the combined actions of the robot team, rather than the individual robot actions. Inherently cooperative tasks are a particular challenge in multi-robot learning due to the difficulty of assigning credit for the individual actions of the robot team members. Credit assignment is difficult in a multi-robot system because the individual robot has only a partial view of the global situation. Effective cooperation requires that the global problem-solving situation influence the local control decisions made by each robot. A robot with a purely local view of the problem solving situation has difficulty learning effective cooperative control decisions because of the uncertainty in its knowledge of the overall state of the system. In a global reinforcement learning paradigm, the only available information is reinforcement that measures the performance of the group during a learning phase. The robot cannot easily determine if the group reward is due to its own action or not. Moreover, several robot actions can have antagonistic effects on the reinforcement function. We note that a better observation of the situation cannot be achieved by using POMDP techniques (Partial Observation Markov Decision Process), since a model of the robot-world interaction is not available.

The application of reinforcement learning [10] is useful in robot control tasks, but it usually requires a well-defined state space by which the robot learns to select an adequate action for each current state to accomplish the task. Typical solutions to the state space definition were proposed in [17, 18], where variable resolution discretization of the input data were used in order to solve control problems. In [5] similar ideas were proposed, but most of these techniques assume a deterministic environment, and are thus not easily applicable to a robotic domain, where indeterminism from sensors and actuators is always present. In this sense, several approaches can be found. In [2], hyper ellipsoids were used to cluster stored vectors and to learn a state transition map in terms of actions from which the optimal action sequence is obtained. In [24], situations are learned and maintained in the continuous state space on the basis of the rewards

from the environment, learning the transitions between the situations in order to apply partial planning over the network learned. In [22], another heuristic is presented to split the state space environment on the basis of the rewards. In general, most of these approaches have been studied in low size domains, typically with only two or three attributes for the input state representation.

Other related work shows the utility of using quantization techniques to represent the state space. An adaptive input-space quantization was proposed in [14] based on Voronoi quantizers, where neurons were added and removed based on the behaviour of the robots. In [6], the Kohonen LVQ algorithm [13] and Q-Learning [25] work together to locate a pre-defined number of neurons in adequate positions of the environment to achieve the goal. In [23], associative memory [12] and Q-Learning are used to learn a map from an input representation of the space state to actions.

In this paper we study the application of the VQQL [8] technique for learning cooperative behaviours in *inherently* cooperative domains, i.e., such domains where the utility of the action of each robot is dependent upon the current actions of the other team members. This technique allows the application of the Q-Learning algorithm over local information to learn behaviours in large domains by obtaining reduced representations of such domains using an unsupervised learning technique [16, 15]. This technique allows a dramatic reduction in the number of states that would be needed if a uniform representation of the environment of the robots were used, thus allowing a better use of the experience obtained from the environment. In this way, useful information can be stored in the state space representation without excessively increasing the number of states needed.

The collaboration among robots emerges automatically from the behaviour of each robot, whose goal is to earn a maximum gain from the environment. This gain is the key point in the learning phase because it implicitly introduces the desired behaviour of the robots.

In this research, the CMOMMT [20] application has been used as a test bed. This test bed is being widely used in distributed robotics research [28, 3] and allows us to make several comparisons with previous works [21].

The next section describes the basis of the VQQL technique and its use in reinforcement learning problems, as well as a brief description of the Generalized Lloyd Algorithm and Q-Learning. Section 3 describes the CMOMMT application, as was defined in [20]. Section 4 presents the main experiments performed and their results. Finally, Section 5 discusses the main conclusions and further research.

2 Reducing the Domain Size

The reinforcement learning model [10] is usually explained in terms of finite and discrete parameters. The usual execution cycle begins with an agent receiving information from the environment. This information is used to represent an individual state s from a finite set $\mathcal{S}$. Given that state, the action policy finds which action a from a finite set $\mathcal{A}$ is useful to achieve a goal. Then, the agent executes the action, receiving feedback from the environment about the new state s' and a reinforcement signal, $r \in \mathcal{R}$. This discrete scheme is useful when time is not continuous and the execution of actions generates state transitions. In this case, the discount parameter γ used for learning is an integer parameter that ranges from 0 to infinity.

However, in most domains, such as the CMOMMT application, none of the previous assumptions about discrete environments are true; we therefore need a mechanism to discretize states and actions in a world with continuous time. Thus, a discretized reinforcement learning problem can be defined as follows [7]:

- The continuous state space, $\mathcal{S}$, is mapped to a finite one, $\hat{\mathcal{S}}$. Each discretized new state can be considered as a region or situation in the continuous state space.

- The continuous action space is mapped to a discretized set of high-level macro-actions or skills, $\hat{\mathcal{A}}$.

- State transitions exist only among regions by executing macro-actions. A state transition from any region $\hat{s}$ to other region $\hat{s'} \in \hat{\mathcal{S}}$ with a macro-action $\hat{a} \in \hat{\mathcal{A}}$ is given when, from any continuous state s in region $\hat{s}$, the agent executes macro-action $\hat{a}$ until it arrives in a continuous state s' into a region $\hat{s'}$ different than $\hat{s}$. That is, a state transition is given only when the agent changes regions in the state space. This transition is given in a continuous time t.

- The reinforcement scheme used is a delayed reward scheme, where positive or negative rewards are obtained only at the end of a training session. Then, if while the agent is in a region it arrives at an end state, it receives an immediate reward from the environment and updates its policy. If the agent arrives in any other region, it receives a null reward and updates its policy taking into account the region in which it arrives and the time that it has taken.

In the next subsections, the clustering technique used to obtain a state space representation is presented, as well as a brief description of the Q-Learning algorithm.

2.1 Generalized Lloyd Algorithm (GLA)

The Generalized Lloyd Algorithm [16, 15] is a clustering technique that consists of a number of iterations, each one recomputing the set of more appropriate partitions of the input states (vectors), and their centroids. The algorithm is shown in Table 1. It takes as input a set T of M input states, and generates as output the set C of N new states (*quantization levels*).

There are three design decisions to be made when using such a technique:

Table 1: The Generalized Lloyd Algorithm.

Generalized Lloyd Algorithm (T, N)

1. Begin with an initial codebook C_1.

2. Repeat:

   (a) Given a codebook (set of clusters defined by their centroids) $C_m = \{y_i; i = 1, \ldots, N\}$, redistribute each vector (state) $x \in T$ into one of the clusters in C_m by selecting the one whose centroid is closer to x (nearest neighbour rule).

   (b) Recompute the centroids for each cluster just created, R, to obtain the new codebook C_{m+1}, using equation (1):

   $$cent(R)[i] = \frac{1}{\|R\|} \sum_{j=1}^{\|R\|} x_j[i] \qquad (1)$$

   where $x_j \in R$, $x_j[i]$ is the value of component (attribute) i of vector x_j, and $\|R\|$ is the cardinality of R.

   (c) If an empty cell (cluster) was generated in the previous step, an alternative code vector assignment is made (instead of the centroid computation).

   (d) Compute the average distortion for C_{m+1}, D_{m+1}.

   Until the distortion has only changed by a small enough amount since last iteration.

Stopping criterion. Usually, the average distortion of a codebook at cycle m, D_m, is computed and compared to a threshold θ ($0 \leq \theta \leq 1$) as in equation (2):

$$(D_m - D_{m+1})/D_m < \theta \tag{2}$$

Empty cells. One of the most used mechanisms consists of splitting other partitions, and reassigning the new partition to the empty one. All empty cells generated by the GLA are changed in each iteration by another cell. To define the new one, another non-empty cell, y, with big average distortion is split in two:

$$y_1 = \{y[1] - \epsilon, \ldots, y[K] - \epsilon\}, \text{and}$$

$$y_2 = \{y[1] + \epsilon, \ldots, y[K] + \epsilon\}$$

Initial codebook generation. We have used a version of the GLA [15] as explained in Table 2, that requires a partition split mechanism as the one described above inserted into the GLA in Table 1.

2.2 Q-learning

Q-learning [25, 26] is one of the most used reinforcement learning algorithms for an agent to learn an action policy. It is based in the use of a state-action table containing the gain that the agent obtains by executing an action from a state. This table represents a function Q that tells the agent which action it should execute in order to obtain a maximum gain. The algorithm is summarized in Table 3.

2.3 Application of *VQ* to *Q-learning*: VQQL

The use of vector quantization and the Generalized Lloyd Algorithm to solve the generalization problem in reinforcement learning algorithms requires two

Table 2: A version of the Generalized Lloyd Algorithm that solves the initial codebook and empty cell problems.

GLA with Splitting (T)

1. Begin with an initial codebook C_1 with N (number of levels of the codebook) set to 1. The only level of the codebook is the centroid of the input.
2. Repeat:
   (a) Set N to $N * 2$
   (b) Generate a new codebook C_{m+1} with N levels that includes the codebook C_m. The rest of the N undefined levels can be initialized to 0.
   (c) Execute the GLA algorithm in Table 1 with the splitting mechanism with parameters (T, N) over the codebook obtained in the previous step.

   Until N is the desired level.

Table 3: Q-learning algorithm.

Q-learning algorithm (S, A)

For each pair ($s \in \mathcal{S}$, $a \in \mathcal{A}$), initialize the table entry $Q(s, a)$ to 0.

Observe the current state s.

Do forever:

- Select an action a and execute it.
- Receive immediate reward r.
- Observe the new state s'.
- Update the table entry for $Q(s, a)$ as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \qquad (3)$$

- Set s to s'.

consecutive phases:

Learn the quantizer. Design the N-levels vector quantizer from input data obtained from the environment.

Learn the Q function. Once the vector quantizer is designed (i.e., the environment is clustered into N different states), the Q function must be learned, generating the Q table composed of N rows and a column for each action.

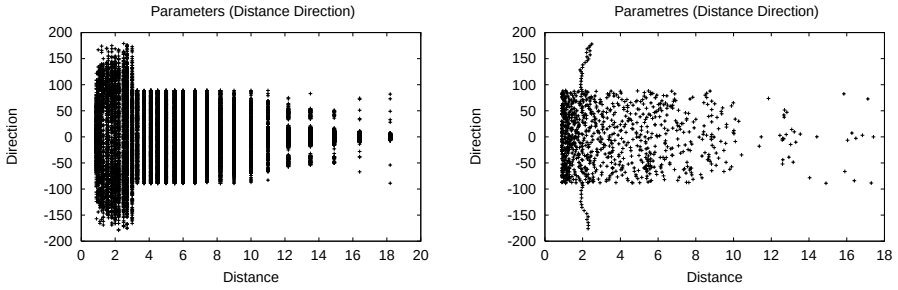
To apply vector quantization with the goal of a domain reduction, followed by the application of a discrete reinforcement learning technique, two strategies can be used: an off-line version where all the initial data is obtained in a previous step, or an on-line version, where the learning phase is interactive, and exploitation strategies can be followed. In this work, the on-line version of the model is applied, using the following steps:

- Obtain a set T of examples of states.
- Design a vector quantizer C using T with the Generalized Lloyd Algorithm.
- Learn the Q function using the following steps:
  - Choose an action following an exploration/exploitation strategy. Receive the reward, obtaining an experience tuple $< s, a, s', r >$. (This tuple is obtained by following the discretized reinforcement learning approach introduced at beginning of this section.)
  - Quantify the experience tuple, obtaining $< \hat{s}, a, \hat{s'}, r >$.
  - Create the Q table following equation (3).

Both phases – the design of the vector quantizer and the reinforcement learning technique – are applied within the VQQL model [8].

2.4 Previous Work

VQQL was first used in the RoboCup domain [11] in the simulation league [19]. The Vector Quantization technique allows us to take advantage of the statistical characteristics of the domain to reduce its size, taking into account only relevant zones from the complete domain. In [8] the ball interception skill of a goalie is learned in the robosoccer domain. Fig. 1 (a) shows examples of the distance and direction parameters from the ball to the goalie – two of the four parameters used for learning the skill. Fig. 1 (b) shows the clusters obtained when the GLA algorithm is used.



(a) (b)

Figure 1: Distance and direction parameters from (a) original data, and (b) a codebook obtained by the GLA.

In this case, data obtained from random behaviour of the agent was used to learn the clusters. After, using Q-Learning to learn the skill, more than 60% of balls are intercepted by the goalie. In Fig. 1, we can see how GLA places clusters only in those zones of the environment where there are examples of the original data, with the bias of situating more clusters in those zones with the highest density of examples.

In [7], an iterative version of the algorithm obtains advantages from the behaviour that the robot is learning in the "Car on the Hill" control task. The main idea is that the state representation might change while the robot is

learning, so it iteratively repeats both phases of the VQQL model: learning the state space representation and learning the behaviour.

3 The CMOMMT Application

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) [20]. In this section, the application is described, as well as previous approaches.

3.1 Definition

The CMOMMT application is defined as follows. Given:

- $\mathcal{S}$: a two-dimensional, bounded, enclosed spatial region
- $\mathcal{V}$: a team of m robot vehicles, $v_i, i = 1, 2, ...m$, with 360^0 field of view observation sensors that are noisy and of limited range
- $\mathcal{O}(t)$: a set of n targets, $o_j(t)$, $j = 1, 2, ..., n$, such that target $o_j(t)$ is located within region $\mathcal{S}$ at time t

We say that 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:

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

Then, the goal is to develop an algorithm, which we call *A-CMOMMT*, 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, we define $sensor\_coverage(v_i)$ as the region visible to robot v_i's observation sensors, for $v_i \in \mathcal{V}$. Then we assume that, 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, $\bigcup_{v_i \in \mathcal{V}} sensor\_coverage(v_i) \ll \mathcal{S}$. 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.

Fig. 2 shows a simulation of the CMOMMT application. In this simulator, small circles represent robots, small squares represent targets, and larger circles around each robot represent its viewing range.

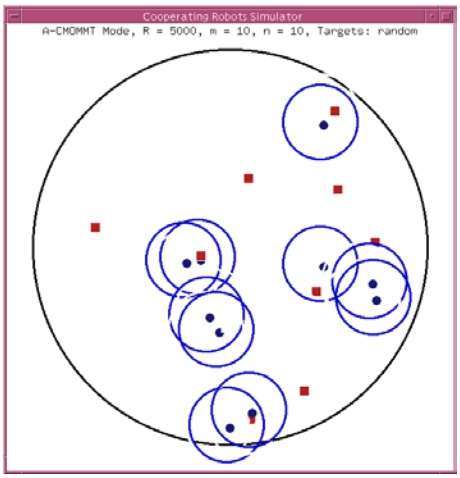


Figure 2: CMOMMT application.

3.2 Previous Approaches

In [21], some results are reported in the CMOMMT application. In this previous work, two main approaches are presented – a hand-generated solution and a learning approach.

The hand-generated solution (called A-CMOMMT) is based on vectors of attraction and repulsion from each robot to the targets and other robots. The result of this 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 in this previous research, called Pessimistic Lazy Q-Learning, is based on a combination of lazy learning [1], 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. Fig. 3 compares the results of the hand-generated solution and the Pessimistic Lazy Q-learning approach, along with two simple control cases – Local and Random – as a function of the size of the memory used.

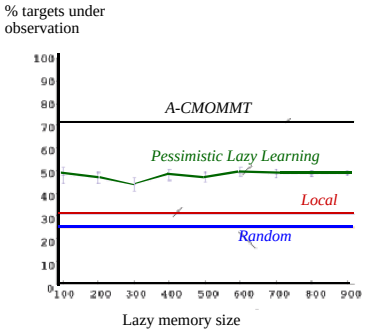


Figure 3: Performance of different approaches to the CMOMMT application.

These results show that the pessimistic algorithm achieves improved results over 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 of [21].

3.3 Reinforcement Learning in the CMOMMT Application

In our current research, the CMOMMT application has been redefined as a delayed reward reinforcement learning problem. To explain this approach, we discuss several issues in the following subsections.

3.3.1 The Input Data.

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, this approach maintains information on the nearest target and the nearest robot, using a mask when information is not known. In this case, the size of the input data depends on the number of targets and robots used as local information, and will differ in different experiments.

3.3.2 The Actions.

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 in a discretized state $\hat{s'} \neq \hat{s}$.

3.3.3 The Reinforcement Function.

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 gained with higher numbers of targets in view. As will be described below, 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

The experiments consist of applying the VQQL model to the CMOMMT application in order to achieve a higher performance, following the performance measure defined in [20] as the average number of targets under observation in runs of 1000 cycles. As in [20], the maximum range for a target or a robot to be seen is 1000, for an arena radius of 5000.

In this case, the on-line version of the model defined in Section 2 has been used. Different experiments have been executed in order to compare the performance of the model based on two main factors – (1) the number of states used for the state space representation, and (2) whether robots use collaboration in order to achieve a higher performance.

4.1 Experiment 1

In the first set of experiments, each 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 of the robot. Fig. 4 shows the x and y components of the distance vector of the input data obtained in the first step of the VQQL model.

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

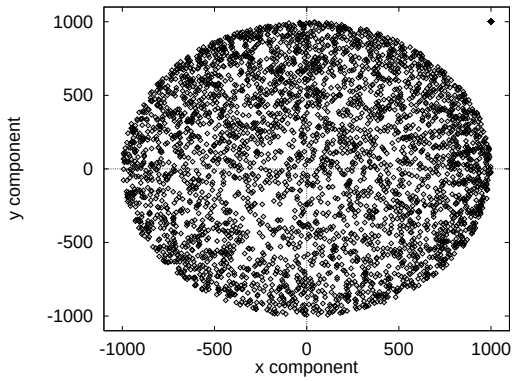


Figure 4: Distance vectors from the robot to the furthest target within viewing range.

point $(1000, 1000)$, which is the mask value used when the robot cannot see any target.

When applying the GLA algorithm to obtain a smaller representation of all this data, we can obtain several codebooks of different sizes. Fig. 5 shows the distortion evolution of the training data and the test data, and how in both cases, the average distortion obtained is less than 1000 for codebooks of 256 states or more.

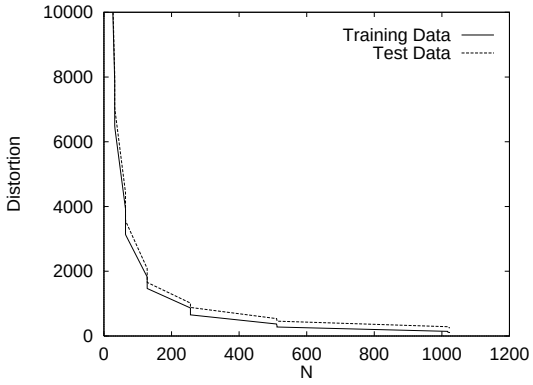


Figure 5: Distortion evolution versus size of the codebook.

Fig. 6 shows the state representation obtained with both a codebook of 16

and a codebook of 64 centroids, i.e., a domain with 16 or 64 different states. We can see how this representation is adapted to the input data shown in Fig. 4, including a state for the point $(1000, 1000)$.

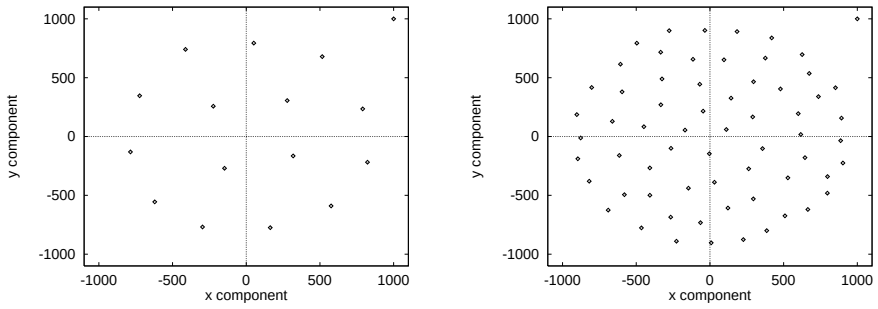


Figure 6: Codebooks representing the state space representation obtained.

Given this representation, the next step is to learn the multi-target observation task. As in the rest of the experiments, eight actions have been defined for the robot, each defined to follow a cardinal point: "go north", "go north-east", "go east", etc. 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 1 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 equal to 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.

Fig. 7 shows the results of the learning for different size codebooks and different learning phase lengths. From this figure, we see that the VQQL model achieves a performance of approximately 59% for most of the studied cases. It is an improvement over previous learning work explained in Section 3, where a 50% rate of performance was achieved with the Pessimistic Lazy Q-Learning

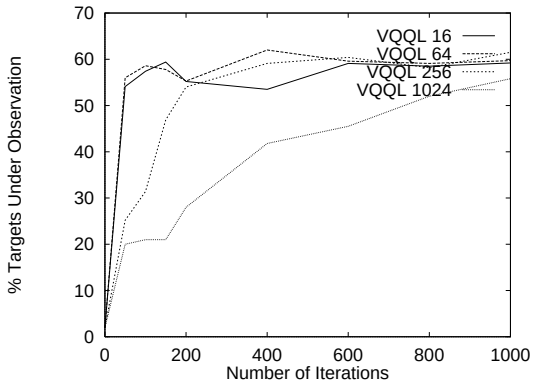


Figure 7: Performance of the VQQL model in the CMOMMT application for various codebook sizes.

algorithm.

In this case, with only 16 different states the robots achieve a 59% rate of performance; higher state representations do not provide higher performance improvements. Another important issue is the learning rate. Smaller state space representations (16 or 64 states) learn very fast, requiring only 100-150 trials, while higher numbers of states (256) need at least 200 trials to achieve similar results. The 1024 level model cannot achieve this success within 1000 trials. Finally, we note that convergence in learning is harder to achieve because of the exploration and exploitation strategy followed in the Q-Learning phase, where the learning rate α is constant (0.05), for a discount parameter of $\gamma = 0.6$. Convergence might be improved with another strategy.

Why can the robots not achieve a higher performance? The answer to this question can be easily found in fig. 8. In this figure, note how ten robots are grouped into only three clusters. In this case, the same performance could be achieved if only three robots were used instead of ten. The reason is that no collaboration strategy is defined, so several robots may be following the same target at the same time.Thus, a collaboration strategy must be introduced in order to avoid this sort of behaviour.

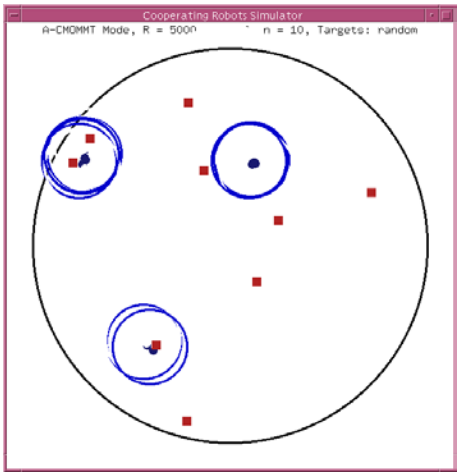


Figure 8: Robots following targets in the CMOMMT application. The behaviour is not correct because several robots follow the same target instead of following different targets.

4.2 Experiment 2

The goal of this experiment is to achieve a better performance by introducing collaborative behaviours among the robots. In this sense, given that the only signal that the robots receive in order to learn their behaviour is the reinforcement signal, the collaboration must be implicitly introduced.

In this case, the state space representation is increased in order to incorporate more information about targets and other robots. Thus, input data is composed of information about the nearest target within view, the furthest target within view, as well as the nearest robot. This increases the input vector from two to six components.

Adding this new information requires a change in the training phase as well, in that now, ten robots simultaneously learn the same policy using the same Q table during learning and testing. The implication of this approach is that the behaviour is learned up to ten times faster, due to ten robots creating the table

simultaneously (although this point has not been studied in depth).

Furthermore, the reinforcement signal now incorporates a negative reward that is given at the end of each trial in order to achieve a collaborative behaviour. 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):

$$r_i(T) = (\sum_{j=1}^{n} b_{ij}(T)) - k(T) \tag{4}$$

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 robots (10 in this experiment),
- and $k(T)$ is a function whose value is 1 if the robot can see other robots, or 0 otherwise.

The basic idea is that the optimal behaviour will be achieved when each robot follows a different target, to the greatest extent possible. This negative reward does not insure this characteristic, but it is easy to understand that it might help.

The results shown in fig. 9 illustrate these prior expectations. In this case, a higher number of states are needed to achieve good behaviour because of the increase in the number of input attributes. For state space representations of only 64, around a 40% rate of performance is achieved. For 256 states, the performance is increased up to 50%, and for 1024 states, the 60% level of performance achieved in experiment 1 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 [21].

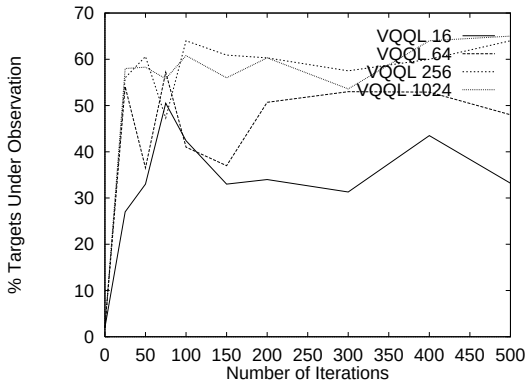


Figure 9: Performance of the VQQL model in the CMOMMT application.

Fig. 10 shows the robots following the targets using the learned behaviour over a state representation of 2048 states. In this image, only one target is not followed by any robot, showing how the collaborative behaviour has been achieved.

Fig. 11 shows the results of both experiments, and their comparison with previous works.

5 Conclusions and Further Research

The primary findings from this research are that the VQQL model is useful for obtaining state space representations that allow the application of the learning model based on discrete representations. In this sense, with only two attributes, the model is able to achieve results of 59% of targets in the robots viewing area with only 16 states – an improvement over previous learning research that achieved a 50% rate of performance. Furthermore, the model is able to learn in a higher dimensional environment, where six attributes were introduced to obtain information about more targets and other robots. In this case, collaborative behaviours emerge only by penalizing robots for maintaining other robots within view, achieving a 65% rate of performance, nearer to the 71% achieved by the

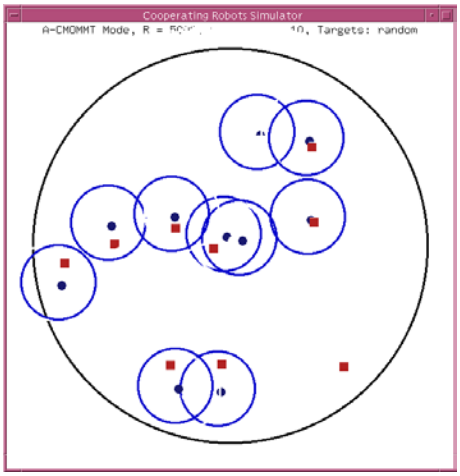


Figure 10: Robots following targets in the CMOMMT application. The behaviour is better than in previous results because most of the targets are followed by only one robot.

best hand-generated solution. In conclusion, collaborative behaviours have been achieved by only using local information of the environment, obtaining better results than with the non-collaborative version.

In future research, primary efforts must be aimed at achieving better performance by obtaining better representations of the state space. In this sense, guiding the unsupervised technique to use information about the behaviour being learned may be useful. Some previous works introduce as information the Q value of the states [23] or the reinforcements received from the environment [24]. On the other hand, the advantages of this approach over the unsupervised techniques are not obvious. The main goal of this technique is to reduce the learning time by not including useless states that lead to inefficiency in learning. However, in some cases, the time and resources spent in learning a more adjusted state space representation might be longer than the penalty for using a less adjusted state representation. Furthermore, unsupervised techniques have been

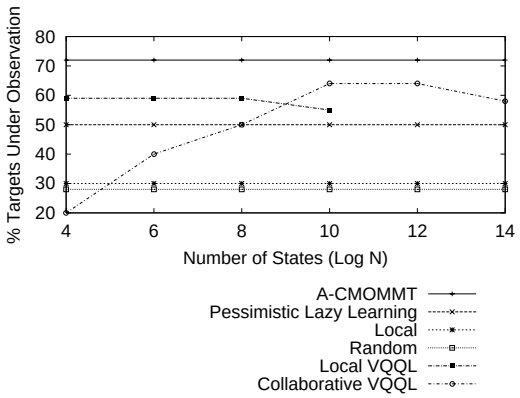


Figure 11: Performance of different approaches to the CMOMMT application

demonstrated to be useful even in classification problems [4, 9].

Acknowledgments

This work has been sponsored in part by the Researcher Staff Formation Program of the Spanish *Ministerio de Educación, Cultura y Deportes* and in part by the Engineering Research Program of the Office of Basic Energy Sciences, U.S. Department of Energy. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. Oak Ridge National Laboratory is managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725.

References

[1] D. Aha, editor. *Lazy Learning.* Kluwer Academic Publishers, 1997.

[2] Minoru Asada, Shoichi Noda, and Koh Hosoda. Action-based sensor space categorization for robot learning. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'96)*, volume 3, pages 1502–1509, 1996.

[3] J. Barhen and V. Protopopescu. Ultrafast neural network training for robot learning from uncertain data. In *Distributed Autonomous Robotic Systems 4*, pages 347–356, 2000.

[4] James C. Bezdek, Thomas R. Rechherzer, Gek Sok Lim, and Yianni Attikiouzel. Multiple-prototype classifier design. *IEEE Transactions on Systems, Man and Cybernetics*, 28(1):67–79, February 1998.

[5] David Chapman and Leslie P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991.

[6] C. Claussen, S. Gutta, and H. Wechsler. Reinforcement learning using funtional approximation for generalization and their application to cart centering and fractal compression. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Coference on Artificial Intelligence*, volume 2, pages 1362–1367, Stockholm, Sweden, August 1999.

[7] Fernando Fernández and Daniel Borrajo. Iterative VQQL for learning skills. In *Proceedings of Learning'00*, Leganés, Madrid, Spain, 2000.

[8] Fernando Fernndez and Daniel Borrajo. VQQL. Applying vector quantization to reinforcement learning. In *RoboCup-99: Robot Soccer World Cup III*. Springer Verlag, 2000.

[9] Bernd Fritzke. Growing cell structures -a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[10] Leslie Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *International Journal of Artificial Intelligence Research*, pages 237–285, 1996.

[11] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Learning Robots*, pages 19–24, December 1995.

[12] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer, Berlin, Heidelberg, 1984. 3rd ed. 1989.

[13] Teuvo Kohonen. Learning vector quantization for pattern recognition. Report TKK-F-A601, Helsinki University of Technology, Espoo, Finland, 1986.

[14] Ben J.A. Krose and Joris W.M. van Dam. Adaptive state space quantization for reinforcement learning of collision-free navigation. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'92)*, volume 2, pages 1327–1332, 1992.

[15] Yoseph Linde, André Buzo, and Robet M. Gray. An algorithm for vector quantizer design. In *IEEE Transactions on Communications, Vol1. Com-28, N 1*, pages 84–95, 1980.

[16] S. P. Lloyd. Least squares quantization in pcm. In *IEEE Transactions on Information Theory*, number 28 in IT, pages 127–135, March 1982.

[17] Andrew W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces. *Proceedings in Eighth International Machine Learning Workshop*, 1991.

[18] Rmi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Coference on Artificial Intelligence*, volume 2, pages 1348–1355, Stockholm, Sweden, August 1999.

[19] Itsuki Noda. Soccer server: a simulator of robocup. In *Proceedings of the Fourth International Symposium'95*, December 1995.

[20] L. E. Parker. A case study for life-long learning and adaptation in cooperative robot teams. In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, pages 92–101, 1999.

[21] Lynne Parker and Claude Touzet. Multi-robot learning in a cooperative observation task. In L.E. Parker, G. Bekey, and J. Barhem, editors, *Distributed Autonomous Robotic Systems*, volume 4, pages 391–401. Springer, 2000.

[22] Tsutomu Sawada, Sumiaki Ichikawa, and Fumio Hara. Autonomous action-mode change in a two-mobile robotics system. s-temperature based on-line learning. In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'99)*, volume 1, pages 393–399, 1999.

[23] Claude Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 1997.

[24] A. Ueno, K. Hori, and S. Nakasuda. Simultaneous learning of situation classification based on rewards and behavior selection based on the situation. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'96)*, volume 3, pages 1510–1517, 1996.

[25] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[26] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[27] Gerhard Weiss and Sandip Sen, editors. *Adaption and Learning in Multi-Agent Systems*. Springer, 1996.

[28] Barry Brian Werger and Maja Matarić. Broadcast of local eligibility for multi-target observation. In L.E. Parker, G. Bekey, and J. Barhem, editors, *Distributed Autonomous Robotic Systems*, volume 4, pages 347–356. Springer, 2000.