A Finite Horizon DEC-POMDP Approach to Multi-robot Task Learning

Barış Eker, Ergin Özkucur, Çetin Meriçli, Tekin Meriçli, and H. Levent Akın

***Abstract*—Decision making under uncertainty is one of the key problems of robotics and this problem is even harder in the multi-agent domain. Decentralized Partially Observable Markov Decision Process (DEC-POMDP) is an approach to model multi-agent decision making problems under uncertainty. There is no efficient exact algorithm to solve these problems since the worst case complexity of the general case has been shown to be NEXP-complete. This paper demonstrates the application of our proposed approximate solution algorithm, which uses evolution strategies, to various DEC-POMDP problems. We show that high level policies can be learned using simplified simulated environments which can readily be transferred to real robots despite having different observation and transition models in the training and the application domains.**

I. Introduction

In order to accomplish its task, an autonomous robot needs to sense the environment and decide on how to act. The state representation of the robot usually considers only the relevant information obtained at a time, however in some cases it may cover the past data as well. In order to be able to decide on how to act in a given state, the robot should have a policy that maps its states to its actions. The difficulty of a decision problem is determined by several factors including the cardinality of the state space, which is defined as the set of all possible states.

In most real world problems, robots can not fully observe the environment and their sensor readings are usually noisy. Such environments are called *partially observable* and Partially Observable Markov Decision Process (POMDP) framework is a generalization of Markov Decision Processes (MDP) used for solving real world problems with the available partial information. In the autonomous robots domain, planning and navigation are some of the problems that can be modeled as POMDPs [1], [2] The aim in POMDP problems is to find a policy that will maximize the expected total reward of a robot for a given horizon, which is the number of actions taken.

Decentralized POMDP (DEC-POMDP) is a generalized version of the POMDP model, aimed to be used for solving planning problems in dynamic environments involving multiple robots. In such problems, the reward is given to the team instead of the individual robots. It has recently been shown that solving DEC-POMDP problems exactly is NEXP-complete, whereas solving POMDP is PSPACE-complete [3]. Therefore, in the last decade, only a few studies on the exact solutions for DEC-POMDP have been made and they were only able solve toy problems with relatively small number of environment states due to NEXP-completeness [4], [5]. Recently, researchers have been trying to come up with approximate solutions to these problems or new subsets of DEC-POMDP problems which can model some real world problems and are easier to solve [6], [7]. Although current approximate solution algorithms are able to solve slightly more complex problems than the exact solution algorithms, they are still far from handling real world problems that involve very large state spaces [8], [9].

Since solving a DEC-POMDP problem means finding a policy that will maximize the expected team reward, these problems can be considered as optimization problems. In this study, we extend the method we developed for solving DEC-POMDP problems using Evolution Strategies (ES) [10] to generate policies for autonomous robots. The original algorithm had a scalability problem since all the policies were encoded in the genes. As a consequence of using neural networks to represent the policies, the number of states of the problems is no longer a constraint. We execute the learned policies on real robots, namely Aldebaran Nao humanoid robots [11] and FESTO Robotino omni-wheeled robots [12] to demonstrate the flexibility of our methodology in terms of learning high level policies in simplified simulated environments and transferring that knowledge to real robots despite having different observation and transition models in the training and the application domains. The results of our experiments show that the robots are able to achieve their tasks even though their perceptions and actions are much noisier compared to the simulation case.

The rest of this paper is organized as follows. Section II provides background information on DEC-POMDPs. Our approach to the problem and applications of our solutions to the real world realizations are elaborated in Section III. Experiment setup and obtained results are explained in Section IV. Section V gives the conclusions and suggestions for future work.

II. Background

The formal definition of DEC-POMDP can be given as a tuple: $< n, S, A, T, \Omega, Obs, R >$ where n is the number of robots, S is a finite set of states, A is the set of joint actions, T is the state transition function, Ω is the set of joint observations, Obs is the observation function, and R is the immediate reward function.

We define A, the set of joint actions, as the cartesian product of $A_i, (i = 1, 2, \ldots, n)$, which is the set of actions available to $Robot_i$. Each $Robot_i$ selects an action from its action set A_i at each time step and the collection of selected actions constitutes

This work is supported by Boğaziçi University Research Fund through project 09M105, and TUBITAK Project 106E172.

The authors are with the Department of Computer Engineering, Boğaziçi University, 34342, Bebek, Istanbul, Turkey
{akin,baris.eker,ergin.ozkucur}@boun.edu.tr
{cetin.mericli,tekin.mericli}@boun.edu.tr

978-1-61284-832-7/11/$26.00 ©2011 IEEE

a joint action. Similarly, Ω, the set of joint observations, is the cartesian product of $\Omega_i, (i = 1,2,\ldots,n)$, which is the set of observations available to $Robot_i$. At any time step the robots receive a joint observation $o = (o_1, o_2, \ldots, o_n)$ from the environment. We assume that $Robot_i$ has access to only o_i and o_i is a member of the Ω_i observation set. The observation function *Obs* specifies the probability of an observation given a joint action and the current state. It also has the Markov property and is stationary. The immediate reward function R is a function of the current state and the joint action taken, and it is basically used to specify the goal of the robots.

The DEC-POMDP model allows us to represent the environment simply as the set of possible states together with the state transition probabilities. Since the robots have uncertain or incomplete information about the states, for each robot we keep a belief state, which is the probability distribution over S, summarizing the robot's knowledge about the current state. Transition from one state to another depends on the joint actions of the robots and the past states; however, the outcomes of the actions are non-deterministic. The transitions are also assumed to be stationary; that is, they do not change over time.

In our work, we deal with "finite horizon" processes. At each time step, each robot selects an action, receives a local observation, and a global reward is given to the team. The aim is to find the policy that will maximize the expected sum of rewards for the given horizon. It is also assumed that the given reward decreases with some discount factor γ and $0 \leq \gamma < 1$.

III. Proposed DEC-POMDP Solution

In our ES implementation, the aim is to find a chromosome to be used by the robots as their policies that will maximize the robot team's expected reward. The application of the learned policy based on DEC-POMDP approach on various real robot platforms is the novel contribution of this study. The stages of our method can be listed as follows.

A. Encoding

In our algorithm, the team policy is represented as a chromosome. For each robot, we have multiple multi-layer perceptron type neural networks, each corresponding to one action. We used the sigmoid function as the activation function. The inputs consist of the state variables. We have a single layer and the number of hidden neurons are determined empirically for each problem. The output is the $Q(s,a)$ value, which is defined to be the expected discounted sum of future payoffs obtained by taking action a from state s and following an optimal policy thereafter. The action to take at particular state is determined by finding the maximum of these values. The chromosome consists of the weight values of all the neural networks. The structure of a neural network and the part of a chromosome that corresponds to this neural network can be seen in Figure 1.

It can easily be seen that the chromosome size is independent of the size of the state space. It is a function of the number of robots and number of available actions. This means that this approach can be used in problems with very large state spaces.

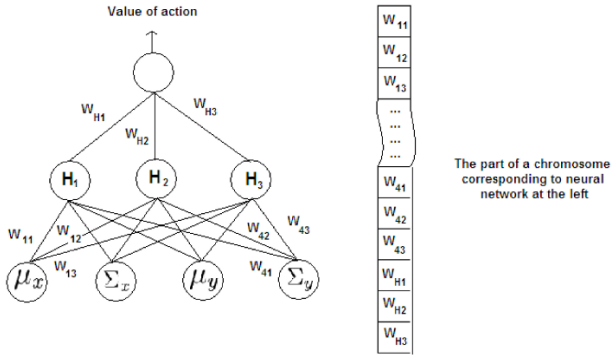

Fig. 1. The neural network and the corresponding part of a chromosome.

B. Fitness Calculation

In DEC-POMDP problems, calculating fitness is a very time consuming task, which is exponential in terms of horizon. Since the fitness is used to sort the chromosomes in our case, the exact reward values are not needed. The aim is to generate as many generations as possible and test their fitnesses in order for our algorithm to converge quickly so that it can be used in larger real life problems. There are several methods to approximate the fitness function; however, no matter which method is used, it is very difficult to obtain a globally correct approximation model. In this study, we use the method given in [10].

C. Convergence Criteria

We determined three possible criteria to terminate evolution.

1) Stop when there is no change
2) Stop when the reward is sufficiently close to optimal
3) Stop after some certain number of generations

The first criterion may fail in cases where the fitness value do not change at the beginning until a policy giving a positive reward is obtained for harder problems. The possible pitfall in using the second criterion may be keeping the algorithm from reaching better solutions by setting a predefined upper limit for the maximum reward value. Therefore, in order to see the progress of evolution for fixed periods and compare several runs better, the algorithm was always run for 2000 generations, which was sufficient for all of our problems.

D. Generating the Next Generation

The population size is empirically set as 30 and the initial set is populated with random chromosomes. The best 10 of them are selected and put in a separate place as potential solutions. These chromosomes do not take place in evolution. If a chromosome having a better fitness value appears during evolution, the worst one is taken out and the new chromosome is put into this set. At the end of 2000 generations, the best chromosomes in this set are tested again in order to find the best one more accurately. The best chromosome obtained corresponds to the team policy. Each robot uses the part that is related with it from this policy; therefore, the individual robot

policies may be different from each other. Recombination and mutation operations are used for creating the next generation.

IV. Experiments & Results

Our experiments mainly consist of two phases; first the policies are learned for the problems at hand in simulation, then they are validated in real world settings using real robots.

A. Learning Policies in Simulation

Several test cases were considered in our experiments where the grid dimensions, available information to the robots, and the level of uncertainty in the robots' actions changed from one test case to another. For each test case, the ES algorithm was run 10 times and 10 different team policies were obtained. In order to obtain a more accurate value for the expected reward of these policies, we get 100000 samples with each of these policies, and report the average reward and the standard deviation of the *best policy* for each test case.

B. Validation of the Learned Policies on Real Robots

We used two Aldebaran Nao humanoid robots [11] and two FESTO Robotino robots [12] to validate the learned policies in a real world setting. The three software modules developed and used for the experiments are *perception*, *planning*, and *motion*, details of which are explained below. We formed 3×3 and 4×3 grids on the floor, each cell being $550mm \times 550mm$. These dimensions are picked in such a way to make sure that the cells are big enough for two robots to fit while the whole grid is small enough to be covered by the overhead camera.

- **Perception:** Although our robots are able to localize themselves in the environment, we decided to use our overhead camera system to track the robots since we surrounded the grid region with cardboard walls and they might cause misperception of the landmarks. We placed unique markers on top of the robots to be able to determine their locations and orientations in the environment (see Figure 2). Using the overhead camera system does not mean that the robots have perfect localization information though. Since the robots are high from the ground and the humanoid robots sway from side to side while walking, their perceived locations by the overhead camera system are not their exact locations; hence noise is present in the location information, which may be up to $200mm$. Another source of noise is the imperfectness of the odometry information provided by the robots. These make the experiment environment definitely partially observable.
- **Motion:** The actions provided in the definitions of grid meeting and U-grid meeting problems are Up, *Down*, $Left$, *Right*, and *Stay*, and *TurnLeft*, *TurnRight*, *MoveForward*, and *Stay* for the cooperative box pushing problem. The directions are the global directions. When the grid world is observed through an overhead camera, the top left corner is denoted as $(0,0)$. In this case *Right* denotes the cell on the right of the current cell, *Down* denotes the cell at the bottom of the current cell, and so on. We made the humanoid robots walk straight between two cells, in order to to move faster. Therefore, when the robot picks an action to perform, the lower level planning module makes the robot first turn towards the direction that it wants to walk in, then the robot walks straight to change its cell.
- **Planning:** The policies learned in the simulation phase are encoded by training a neural network for each action of each robot. For the grid meeting and U-grid meeting problems, each neural network takes the 2D coordinates of the cell the robot resides in as input, and for the cooperative box pushing problem, the neural networks take 5 observations of the robots as input, and they give the $Q(s,a)$ value as the output. When each of the neural networks are fed with the same information, the action that corresponds to the network that gives the highest value is performed since that would be the action resulting in the highest reward.

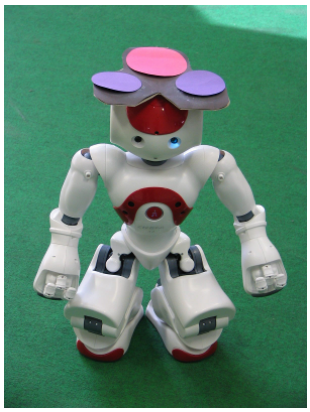

Fig. 2. The Nao humanoid robot with a special marker placed on top of its head.

C. Problems Considered

The problems we considered were the grid meeting problem, the U-grid meeting problem, and the cooperative box pushing problem [10]. Below we present the results for both the simulation and real world validation phases for each problem.

1) The Grid Meeting Problem: The goal of this experiment is to have the robots learn how to meet in the 3×3 grid world regardless of their initial locations. In this problem there are 81 states. As we expected, the robots obtain better reward values when they know their initial states completely. When the robots know their initial locations and their actions are deterministic.

At the end of the training period run in simulation, the robots learn the policy illustrated in Figure 3.

In this problem, the robots start from the top left and bottom right corners, and meet inside the top right cell. The initial and final states of the robots are shown in Figure 4.

Figure 5 shows the path followed by each robot recorded via the overhead camera. The noise in the location information reported by the overhead camera system can be observed in

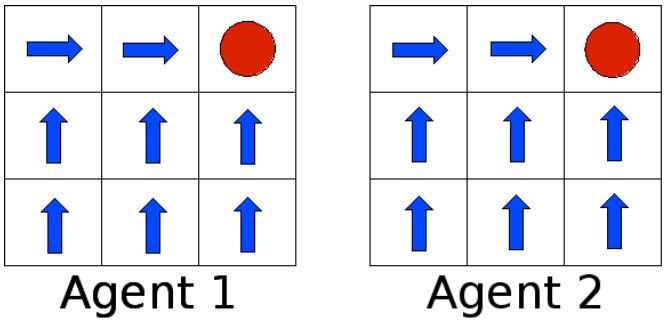

Fig. 3. Learned policy for the grid meeting problem.

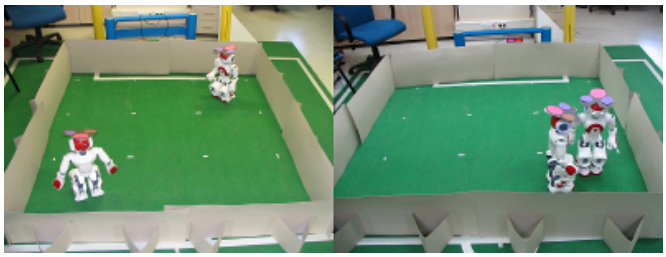

Fig. 4. Start states of the robots and the final state when the robots meet.

this figure. We observed that, regardless of where they start, the execution of the learned policy by the neural network based planning module made the robots successfully walk to the top right cell, where they had met most of the time during the training period, and wait there.

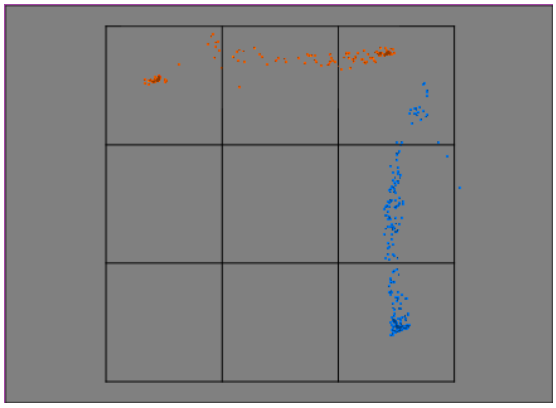

Fig. 5. Paths followed by the two robots during the execution of the learned policies for the grid meeting problem.

2) The U-Grid Meeting Problem: The U-grid meeting problem is a modified version of the grid-meeting problem, where the robots can only move over the outermost cells of the grid, except the topmost part. The robots can not meet in the U-grid meeting problem by always performing the same action, as it is possible in the grid-meeting problem. Hence, they need to learn a more complex policy.

We ran the experiments on a 3×3 grid and as expected, lower reward values are obtained for this problem, because in the optimal case, the robots need to perform an additional action in this problem. As in the grid meeting problem, the optimal reward value is obtained when the robots know the initial state and the actions are deterministic.

The learned policy after the training runs in simulation is illustrated in Figure 6. As was the case for the grid meeting experiment, we again observed the robots performing the learned policy successfully in real world.

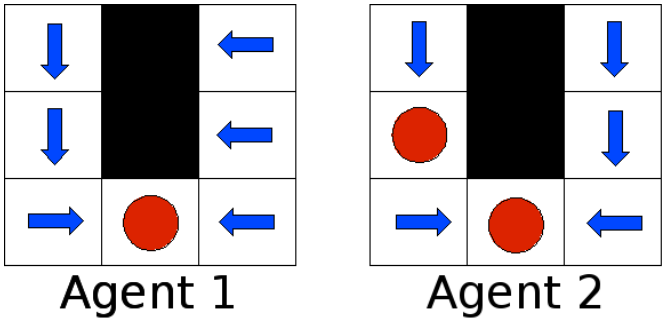

Fig. 6. Learned policy for U-grid meeting problem.

Figure 7 shows the initial and final states of the robots in the U-grid world setting while Figure 8 illustrates the paths followed by the robots during the execution of the learned policy. In this experiment, the robots started from the top left and top right cells, and met inside the bottom middle cell as shown in the figures. The black region in Figure 8 indicates the blocked cells.

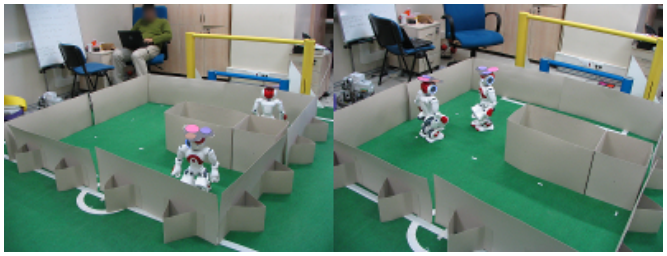

Fig. 7. Start states of the robots and the final state when the robots meet.

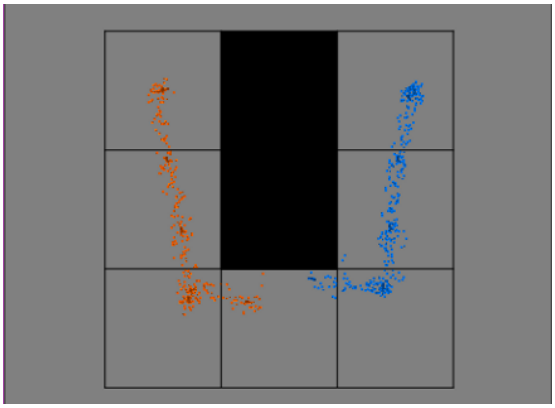

Fig. 8. Paths followed by the two robots during the execution of the learned policies for the U-grid meeting problem.

3) Cooperative Box Pushing Problem: In the original problem definition, there are two types of boxes in the environment and the agents aim to move them to the target area. Small boxes can be pushed by a single robot, however large boxes need two robots to be pushed. The robot gets a deterministic observation from the environment describing the situation in front of it. The possible observations are: *empty field*, *wall*, *other agent*, *small box*, and *large box*. The robots get negative reward for spending too much time, bumping into a wall, or not being able to push a box, and positive reward for being able to push the boxes to the target area. In our experiments, in order to simplify the setup we removed the small boxes from the environment.

Figure 9 shows the initial and final positions of the robots in cooperative box pushing setting. The paths followed by the robots during execution are illustrated in Figure 10.

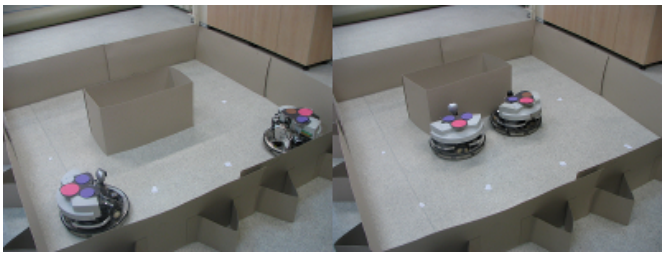

Fig. 9. Start states of the robots and the final state when pushing is completed.

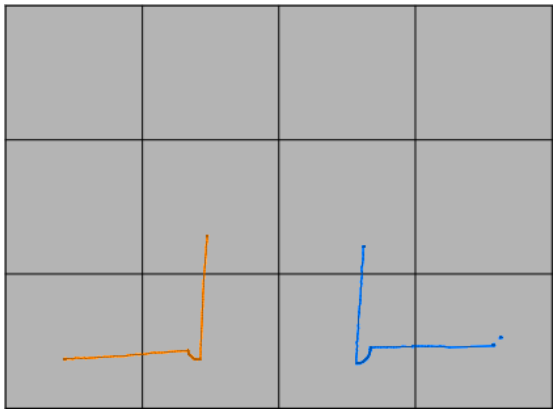

Fig. 10. Paths followed by the two robots during the execution of the learned policies for the cooperative box pushing problem.

V. Conclusions and Future Work

We developed a new algorithm for approximately solving DEC-POMDP problems using evolution strategies and verified the solutions in real world using real robots. It has been shown that this algorithm can find good quality approximate solutions for DEC-POMDP problems and it is also scalable. Our method can be used to solve much larger problems than reported before. Since the ES algorithm searches for a policy, the algorithm can be modified to continue until the user agrees that a good quality solution is found or the search can continue as long as the time permits. These are among the advantages of using ES for solving DEC-POMDP problems.

The results of our experiments demonstrate that the policies that are learned in simplified simulated environments can successfully be used in complex and noisy real world environments. That may be a sign for the possibility of learning complex tasks like playing soccer and collaborative search and rescue in simplified simulation environments in relatively less time rather than trying to learn it in real world or in realistic simulation environments, which requires a significant amount of time. Observing that motivated us for applying the developed method to robot soccer as the first step by learning the necessary skills in simplified simulated environments and using the learned policies on real robots. Therefore, as a future work we plan to use this approach to generate policies for multi-robot teams in real environments.

References

[1] E. Lòpez, R. Barea, L. M. Bergasa, and M. Escudero, "Visually augmented POMDP for indoor robot navigation," in *Applied Informatics*, 2003, pp. 183–187.

[2] M. T. J. Spaan and N. A. Vlassis, "A point-based POMDP algorithm for robot planning," in *ICRA*, 2004, pp. 2399–2404.

[3] D. S. Bernstein, S. Zilberstein, and N. Immerman, "The complexity of decentralized control of markov decision processes," in *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 32–37. [Online]. Available: http://portal.acm.org/citation.cfm?id=719912

[4] E. A. Hansen, "Dynamic programming for partially observable stochastic games," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004, pp. 709–715.

[5] D. Szer and F. Charpillet, "MAA*: A heuristic search algorithm for solving decentralized pomdps," in *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 2005, pp. 576–583.

[6] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman, "Solving transition independent decentralized markov decision processes," *Journal of Artificial Intelligence Research*, vol. 22, pp. 423–455, December 2004. [Online]. Available: http://mas.cs.umass.edu/paper/368

[7] C. V. Goldman and S. Zilberstein, "Decentralized control of cooperative systems: Categorization and complexity analysis," *Journal of Artificial Intelligence Research*, vol. 22, p. 2004, 2004.

[8] D. S. Bernstein, "Bounded policy iteration for decentralized pomdps," in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 2005, pp. 1287–1292.

[9] R. Nair and M. Tambe, "Taming decentralized pomdps: Towards efficient policy computation for multiagent settings," in *IJCAI*, 2003, pp. 705–711.

[10] B. Eker and H. L. Akin, "Using evolution strategies to solve DEC-POMDP problems," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 14, pp. 35–47, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00500-008-0388-7

[11] "Aldebaran-Nao humanoid robot," http://www.aldebaran-robotics.com/eng/Nao.php.

[12] "FESTO Robotino," http://www.festo-didactic.com/int-en/learning-systems/education-and-research-robots-robotino/.