Solving Multi-agent Decision Problems Modeled as Dec-POMDP: A Robot Soccer Case Study

Okan Aşık and H. Levent Akın

Boğaziçi University, Department of Computer Engineering, 34342, İstanbul, Turkey

Abstract. Robot soccer is one of the major domains for studying the coordination of multi-robot teams. Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is a recent mathematical framework which has been used to model multi-agent coordination. In this work, we model simple robot soccer as Dec-POMDP and solve it using an algorithm which is based on the approach detailed in [1]. This algorithm uses finite state controllers to represent policies and searches the policy space with genetic algorithms. We use the *TeamBots* simulation environment. We use score difference of a game as a fitness and try to estimate it by running many simulations. We show that it is possible to model a robot soccer game as a Dec-POMDP and achieve satisfactory results. The trained policy wins almost all of the games against the standard *TeamBots* teams, and a reinforcement learning based team developed elsewhere.

Keywords: DEC-POMDP, genetic algorithms, robot soccer, simulation, high-level planning.

1 Introduction

Robots are physical agents which interact with their environment via their sensors and actuators. The main problem of a robot is finding a method to map its sensor inputs to actuator outputs to achieve its designated goal. This can be modeled as a decision making problem. There are many methods to solve decision making problems. Approaches based on *Markov Decision Process* (MDP) models are widely used compared to other methods.

There are some tasks which require the cooperation of agents, such as robot soccer. All robots act autonomously, but they should be coordinated. Decision making is a more complicated problem for such multi-robot situations because individual actions of the robots should result in the completion of the task of the team, such as scoring. Decentralized Partially Observable Markov Decision Process (Dec-POMDP) model is one of the promising approaches to solve multi-agent decision making under uncertainty. There are different formalizations for Dec-POMDP, in our study we use Bernstein's model [2].

In this paper, we model robot soccer as a Dec-POMDP problem and use the *GA-FSC* algorithm in [1]. The algorithm represents policies as finite-state controllers and searches the policy space with genetic algorithms. We use *TeamBots* [3] 2D robot soccer simulator as the simulation environment. We show that it is possible to develop a successful team that defeats all the predefined teams in the *TeamBots* environment and also a reinforcement learning based team developed in another study [4].

The organization of the rest of the paper is as follows. Section 2 introduces related work. Section 3 presents the algorithm we used to solve Dec-POMDP problem. Section 4 introduces our experiments and results. We present our conclusions and intended future work in Section 5.

2 Related Work

We can categorize Dec-POMDP algorithms as exact and approximate algorithms. Optimally solving Dec-POMDP problems, have been shown to be NEXP-complete [5]. Therefore, exact solutions are not feasible for almost all real-world applications, and the current research is mainly about finding approximate solutions. The algorithms developed so far are generally tested on benchmark Dec-POMDP problems such as Dec-Tiger, multi-access broadcast channel, meeting in a grid, box pushing, and fire fighting problems [1]. They are used to compare and contrast the performances of different algorithms.

Wu and Chen solves the soccer problem modeled as a Dec-POMDP with Correlation-MDPs in the RoboCup domain [6]. They base their work on the memory-bounded dynamic programming algorithm proposed by Bernstein *et al* [2]. Their main contribution is proposing an approximate algorithm to calculate the correlation device. They used the algorithm to improve the coordination of soccer playing agents in the RoboCup 2006 Soccer 2D Simulation Competitions, and they won all the matches except one. This study is important in terms of showing the capabilities of the Dec-POMDP framework in the robot soccer domain.

Keepaway soccer was put forth as a testbed for machine learning [7], and there is a wide variety of reinforcement algorithms which are tested with keepaway soccer [8, 9, 10, 11]. Di Pietro *et al* used evolutionary algorithms to learn a policy which results in coordinated behavior [12]. They formulate the problem so that the agent decisions are based on parameters such as the distance to the recipient. The evolutionary algorithm searches for the optimal parameters to keep the ball as long as possible which is the ultimate goal of keepaway soccer. This work is close to our work in terms of using an evolutionary algorithm and trying to solve the soccer problem, but their solution is problem specific which is a sub-problem of robot soccer.

Although there are many studies on how to learn to play soccer, they have either combined their solution with the existing planning framework or solved a subset of soccer problem such as keepaway soccer [7, 13]. In this paper, we model robot soccer as a Dec-POMDP and represent the policy as a finite state controller. The robots execute the trained policy represented as finite state controllers throughout the game.

3 Solving Problems Modeled as Decentralized Markov Decision Processes

The *Decentralized Partially Observable Markov Decision Process (DEC-POMDP)* [5] model consists of 7-tuple $(n, S, A, T, \Omega, Obs, R)$ where:

- n is the number of agents.
- S is a finite set of states.
- A is the set of joint actions which is the Cartesian product of A_i ($i = 1, 2, \dots, n$) i.e. the set of actions available to $agent_i$.
- T is the state transition function which determines the probabilities of the possible next states given the current state S and the current joint action a .
- Ω is the set of joint observations which is the Cartesian product of Ω_i ($i = 1, 2, \dots, n$) i.e. the set of observations available to $agent_i$. At any time step the agents receive a joint observation $o = (o_1, o_2, \dots, o_n)$ from the environment.
- Obs is the observation function which specifies the probability of receiving the joint observation o given the current state S and the current joint action a .
- R is the immediate reward function specifying the reward taken by the multiagent team given the current state and the joint action.

3.1 Dec-POMDP Policies and Finite State Controllers

A Dec-POMDP policy is a mapping of the observation history to the actions. Generally, policies are represented as a policy tree where observations lead to actions. However, the tree representation is not sufficiently compact. The Finite state controller (*FSC*) representation is one of the viable candidates to represent policies. A *FSC* is a special finite state machine. It consists of a set of states and transitions. The main difference here is that those states called *FSC* nodes, and are abstract and different from the environment states. Every *FSC* node corresponds to one action which is the best action for that particular state. Transitions take place when a particular observation is taken at a particular *FSC* node. An example finite state controller can be seen in Figure 1. This finite state controller is designed for a problem having only two observations and three actions. In a *FSC*, there is always a starting state. Let us assume that the starting state is $S1$ so that $A1$ is executed first. If the robot gets an observation $O2$, it updates its current *FSC* node to $S2$ and executes the action $A2$. Action execution and *FSC* node update continues until the the end of the episode. This finite state controller represents the policy of a single robot. The critical point about the finite state controller representation is that we can model a Dec-POMDP policy with different numbers of nodes. Since every node corresponds to one action, the minimum number of nodes is the number of actions. Since having greater number of nodes than the number of actions does not improve the performance of the algorithm[1], in our experiments, the number of *FSC* nodes is equal to the number of actions.

3.2 Genetic Algorithms

In genetic algorithms, a candidate solution is encoded in a chromosome and the set of all chromosomes is called a *population*. The fitness of a candidate solution determines how good the candidate is. Through the application of evolutionary operators such as selection, crossover, and mutation, a new population is created from the current population. When the convergence criteria are met, the algorithm terminates and the best candidate becomes the solution of the algorithm [14].

Fig. 1. An Example Finite State Controller

Encoding. In order to solve a Dec-POMDP using genetic algorithms, we should encode the candidate solution, the policy. In this study, the encoding of a *FSC* as a chromosome is as follows: the first n genes represent node-action mapping and their values are between 1 and the number of actions (A). Then, for each node, there is an observation-node mapping which denotes the transition when an observation is taken as seen in Figure 2. The value of this range is between 1 and S which represents the number of nodes. The whole chromosome of the Dec-POMDP policy is constructed by concatenating every robot's policy.

A1	A2	A3	S3	S2	S3	S1	S2	S1
Best Action at S1	Best Action at S2	Best Action at S3	O1 is taken at S1	O2 is taken at S1	O1 is taken at S2	O2 is taken at S2	O1 is taken at S3	O2 is taken at S3

Fig. 2. An Example *FSC* Encoding

Fitness Calculation. Fitness calculation is one of the most critical parts of any genetic algorithm. For Dec-POMDP problems for which transition and reward functions can be stated, it is possible to calculate fitness values for a given policy. However, for problems with unknown transition and reward functions, only approximate fitness calculation is possible.

One method of calculating fitness approximately is by running a large number of simulations with a given policy. The fitness of a policy have been shown to stabilize after 1000 simulations for Dec-POMDP benchmark problems [1]. However, for a stable fitness calculation, we should run as many simulations as possible, but the reasonable number of simulations is highly problem dependent. There is a trade-off between the precision of the calculation and the running time complexity of the calculation. One of

the most important factors that have an effect on choosing the number of simulations is accuracy. We need to estimate the fitness value sufficiently accurately so that the chromosomes can be ranked.

3.3 The GA-*FSC* Algorithm

Even though an evolutionary strategy based approach has been proposed in [15], it has been shown to be not sufficiently scalable with the number of agents. In [1] it has been shown that the finite state controller based approach performs better than the previous approach in [15]. For this reason we use the genetic algorithms based approach proposed in [1].

This algorithm has two major components :

- **Encoding the candidate policy:** A policy is represented as a *FSC* and is encoded as an integer chromosome whose details will be given below.
- **Searching the policy space for the best policy with genetic algorithm:** In [1], two fitness calculation approaches are proposed: exact and approximate. For the robot soccer problem considered here, exact calculation is not possible since the dynamics of the environment are not known exactly. The approximate calculation method, however, relies on running many simulations with a given policy and taking the average reward of those simulations as the fitness of the policy.

The algorithm has three stages: pre-evolution, during evolution, and post-evolution. After a random population is formed, the k best chromosomes are selected based on their fitnesses. Those k chromosomes are copied to the best chromosomes list. At the end of each generation, the best k chromosomes of the population are compared to the chromosomes in the best chromosomes list, if it one of the best chromosomes of this generation is better than one of the current best chromosomes, its fitness is calculated more precisely by running additional simulations. If it is still better, it is added to the current best chromosomes list. At the end of the evolution which is determined by setting a maximum generation number, the best of best chromosomes list is determined by running additional simulations. In this study, we keep 10 chromosomes in the best chromosomes list.

3.4 Robot Soccer Dec-POMDP Model

We use the *TeamBots* simulation environment [3] as a testbed for our Dec-POMDP algorithm. The model is directly related to the simulation environment. Different models are required for different simulation environments. Since we have already used *TeamBots* simulation in different studies, we have a well-established MDP model. To model the robot soccer as Dec-POMDP model, we need to define the set of actions, set of observations and the number of states. The finite set of actions is as follows:

$$A = \{Go\ to\ ball, Go\ to\ support\ position, Go\ to\ defense\ position, Pass\ to\ the\ closest\ teammate, Pass\ to\ the\ teammate\ closest\ to\ the\ opponent\ goal\}$$

The finite set of observations is as follows: The *TeamBots* field is divided with 2 equally spaced lines from the narrow edge and 3 equally spaced lines from the wide edge. In total there are 12 grid cells as seen in Figure 3. The *Location* information is based on this grid.

Fig. 3. *TeamBots* Field

We define two observation metrics in those grid cells. The first observation metric called *Dominance* has three possible values based on the number of players in the cell the ball resides:

- *Equal number of players,*
- *The opponent team has more players, and*
- *Our team has more players.*

The other observation metric is called *Closeness*. It also has three possible values which are based on which player is the closest to the ball:

- *An opponent player is the closest,*
- *A teammate is the closest, and*
- *The robot itself is the closest.*

Therefore, the observation set includes three critical pieces of information about the environment: The location of the the ball in the grid, the player the closest to the ball, and the team which is the dominant one in the cell where the ball resides.

$$\text{Observation} = \text{Location} \times \text{Closeness} \times \text{Dominance}$$

4 Experiments and Results

All the experiments in this study are done with the *TeamBots* simulation environment using the *JGAP* genetic algorithms package [16]. In the standard *TeamBots* package there are four standard teams. They are in the order of increasing power: *BrianTeam*,

Kechze, *SibHeteroG*, *AIKHomoG*. In addition there is a team called *NullTeam* which is used for learning very basic behaviors such as dribbling the ball. The players of the *NullTeam* are immobile during the game. The matches are played with teams of 5 players.

We train against all teams iteratively starting from the easiest team up to the hardest team. Our ultimate goal is to fine tune the algorithm so that it is best suited for solving the robot soccer problem modeled as a Dec-POMDP. Since we need a stable fitness calculation, the number of simulations used for estimating the fitness of a candidate policy is one of the parameters we need to determine.

4.1 Genetic Algorithm

When we define our problem as a Dec-POMDP and use *GA-FSC* as a solver, the quality of the solution is highly dependent on the parameters of the genetic algorithm. We determined the genetic algorithm parameters shown in Table 1 empirically.

Table 1. Parameters of the Genetic Algorithm

Parameter	Value
Population Size	50
Mutation Rate	0.1
Crossover Rate	0.5
N_B : Number of Simulations Before Evolution	100
N_D : Number of Simulations During Evolution	50
N_A : Number of Simulations After Evolution	500
Fitness Metric	Score
Maximum Number of Generations	50
Convergence Limit	20

The evolution cycle for training the Dec-POMDP team against a selected standard team is as follows. The first population is initialized randomly. Then, we determine the best chromosomes of the evolution by running N_B simulations. In each generation, we determine the fitness of the chromosomes in the population by running N_D simulations. At the end of every generation, we get the top 10 chromosomes of the population and recalculate their fitness by running N_B simulations. If any one of them is still good enough to be in the best chromosomes list, it is added to the list and the evolution continues. As the termination criteria we use reaching the maximum number of generations or the maximum fitness not changing for a specified number of generations. When the evolution ends we calculate the best solution from the best chromosomes list by running N_A simulations.

Training is carried out in stages. We first train against the *NullTeam*, then against the other standard *TeamBots* teams, in the order of increasing difficulty. The population of a previous team is used for the next team except the *NullTeam* whose population is randomly initialized.

4.2 Fitness Calculation

The main problem about the fitness calculation is that we try to estimate the fitness of a policy by taking many simulation runs. Therefore, we need to find the number of simulation runs which is enough to rank the chromosomes so that the genetic algorithm can converge. In Figure 4, we show the change in the rank of 50 chromosomes over the number of simulations. The change in rank is calculated by summing the change of all chromosomes between two consecutive runs. It is found that 50 simulation runs are enough to distinguish good solution candidate since after 50 simulations the rank of chromosomes do not oscillate much. However, we need to determine two more numbers for simulation runs to achieve higher precision when deciding whether the policy is good enough to be kept as one of the best solutions, and when deciding what is the best of all best candidates. By considering running time limitations, we choose 100 simulation runs to decide whether a policy is good enough to be in the best chromosome list, and we choose 500 simulation runs to decide what is the best solution of best chromosomes list.

Fig. 4. The Change in the Rank of Chromosomes by the Number of Simulations

In robot soccer, the fitness of a policy can be calculated in different ways. One of the possible fitness calculation methods is the score difference. However, score difference may not be a good method since it may not be selective enough to differentiate a good soccer player policy from a bad one when their score is the same. When policies are randomly initialized, none of the policies in the population scores goals against the good teams so that they all have the same fitness. We know that some policies are more successful at playing soccer, but they cannot score. Those chromosomes should be selected for next generations. Therefore, to solve this problem, we train policies iteratively starting with the weaker teams and continuing with the stronger teams. The performance of the method can be seen in Table 2.

Table 2. The Performance of Iterative Training with Score Difference Fitness Method

Opponent	Average Score Difference of 500 Evaluation Runs	Average Score Difference at The End of Evolution for That Team	Best Score Difference	Win	Draw	Loss
NullTeam	8.42	43.96	19	499	1	0
BrianTeam	7.04	22.9	13	500	0	0
Kechze	3.68	4.97	9	493	7	0
SibHeteroG	1.31	1.74	4	399	90	11
AIKHomoG	2.48	3.77	7	460	37	3
Merikli <i>et al</i> team (RL-Based)	1.74	N.A.	6	421	78	1

The difference between the average scores at the end of evolution and the average scores of 500 evaluation runs is high for weak teams such as *NullTeam*, and *BrianTeam*. Since the policies trained against those teams easily converge to successful policies which are a series of simple actions, the score of the evaluation run is lower than the score at end of evolution for that team. Another reason for this difference is that the final best policy is highly adapted to the last teams it is trained against.

One of the most important performance measures for the algorithm is the number of wins and losses. As it is seen in Table 2, the trained policy never loses against *NullTeam*, *BrianTeam*, *Kechze*, and loses only 11 games against *SibHeteroG*, 3 games against *AIKHomoG* out of 500 games. Although, the average score difference against *SibHeteroG*, and *AIKHomoG* is not very high, the number of wins are quite satisfactory.

In addition to the standard *TeamBots* teams, we also report the average scores against the team trained by Merikli *et al* [4]. Even though our team was trained only against the *TeamBots* teams we have a positive average score against the Merikli *et al* team and we win most of the games as seen in Table 2.

4.3 Evaluation of DEC-POMDP Policies

Although there is no benchmark for the *TeamBots* simulation environment, in order to assess the performance of our method we compare our average score with the scores reported in [4]. Although the focus of the work reported in [4] is different from our work, both studies use the same MDP model and the simulation environment, i.e., the same basic actions, state definition, and observation definition. They use the reinforcement learning approach with soccer metrics developed by Merikli *et al* [17]. In Table 3, we compare our results with the scores reported in [4]. Although, our average scores are lower, we achieve positive average scores against all teams and win most of the games against *SibHeteroG*. However, the reinforcement learning based team has a negative average score against *SibHeteroG*.

Table 3. The Comparison of Average Scores

Opponent Team	Average Scores of Dec-POMDP Based Approach	Average Scores of Reinforcement Learning Based Approach [4]
NullTeam	8.42	28.25
BrianTeam	7.04	17.80
Kechze	3.68	12.67
SibHeteroG	1.31	-4.90
AIKHomoG	2.48	N.A.

5 Conclusions

Robot soccer is one of the best testbeds for studying a variety of different techniques in the multi-robot domain. In this paper, we propose the application of a Dec-POMDP algorithm for developing team strategies for robot soccer. We implemented the algorithm in the *TeamBots* 2D simulator and compared the results with the previous work. We found that the algorithm is quite suitable for solving robot soccer decision problems since we get positive average scores against teams of different strength and win almost all of the matches. Another contribution of the study is that we investigated different parameters of the proposed algorithm and their effect to the performance of the solution.

One of the most important limitations of this algorithm is the estimation of the fitness of individual chromosomes. Since it is based on repeating the simulation many times, as the fidelity of the simulator increases, the running time of the simulator also increases. Therefore, we need to deal with a trade-off between the running time, and the accuracy.

In future work, we plan to develop a better fitness evaluation method and experiment with it in the RoboCup 2D simulator. Our ultimate future plan is to implement and experiment this algorithm in the RoboCup 3D simulator and use it in the RoboCup Standard Platform League.

Acknowledgments. This study was supported by Boğaziçi University Research Fund project 09M105.

References

- [1] Eker, B.: Evolutionary Algorithms for Solving DEC-POMDP Problems. PhD thesis, Boğaziçi University (2012)
- [2] Bernstein, D.S., Hansen, E.A., Zilberstein, S.: Bounded Policy Iteration for Decentralized POMDPs. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, pp. 1287–1292 (2005)
- [3] Balch, T.: Teambots mobile robot simulator (2000)
- [4] Meriçli, Ç., Meriçli, T., Levent Akın, H.: A Reward Function Generation Method Using Genetic Algorithms: A Robot Soccer Case Study. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010, Richland, SC, vol. 1, pp. 1513–1514 (2010); International Foundation for Autonomous Agents and Multiagent Systems

- [5] Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.* 27, 819–840 (2002)
- [6] Wu, F., Chen, X.: Solving Large-Scale and Sparse-Reward DEC-POMDPs with Correlation-MDPs. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) *RoboCup 2007. LNCS (LNAI)*, vol. 5001, pp. 208–219. Springer, Heidelberg (2008)
- [7] Stone, P., Sutton, R.S.: Scaling Reinforcement Learning toward RoboCup Soccer. In: *Proc. 18th International Conf. on Machine Learning*, pp. 537–544. Morgan Kaufmann, San Francisco (2001)
- [8] Stone, P., Sutton, R.S., Singh, S.: Reinforcement Learning for 3 vs. 2 Keepaway. In: Stone, P., Balch, T., Kraetzschmar, G.K. (eds.) *RoboCup 2000. LNCS (LNAI)*, vol. 2019, pp. 249–258. Springer, Heidelberg (2001)
- [9] Stone, P., Sutton, R.S., Singh, S.: Reinforcement Learning for 3 vs. 2 Keepaway. In: Stone, P., Balch, T., Kraetzschmar, G.K. (eds.) *RoboCup 2000. LNCS (LNAI)*, vol. 2019, pp. 249–258. Springer, Heidelberg (2001)
- [10] Whiteson, S., Kohl, N., Mikkiläinen, R., Stone, P.: Evolving Soccer Keepaway Players Through Task Decomposition. *Machine Learning* 59, 5–30 (2005), 10.1007/s10994-005-0460-9
- [11] Stone, P., Kuhlmann, G., Taylor, M.E., Liu, Y.: Keepaway Soccer: From Machine Learning Testbed to Benchmark. In: Bredendfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) *RoboCup 2005. LNCS (LNAI)*, vol. 4020, pp. 93–105. Springer, Heidelberg (2006)
- [12] Pietro, A.D., While, L., Barone, L.: Learning In RoboCup Keepaway Using Evolutionary Algorithms. In: *GECCO 2002*, pp. 1065–1072 (2002)
- [13] Amato, C., Bernstein, D.S., Zilberstein, S.: Optimal Fixed-Size Controllers for Decentralized POMDPs. In: *Proceedings of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, Hakodate, Japan, pp. 61–71 (2006)
- [14] Levent Akin, H.: Evolutionary Computation: A Natural Answer to Artificial Questions. In: *Proceedings of ANNAL: Hints from Life to Artificial Intelligence*, pp. 41–52. METU, Ankara (1994)
- [15] Eker, B., Levent Akin, H.: Using evolution strategies to solve DEC-POMDP problems. *Soft Computing-A Fusion of Foundations, Methodologies and Applications* 14(1), 35–47 (2010)
- [16] Meffert, K., Meseguer, J., Marti, E.D., Meskauskas, A., Vos, J., Rotstan, N.: Jgap: Java genetic algorithms package (2011)
- [17] Meriçli, Ç., Levent Akin, H.: A Layered Metric Definition and Evaluation Framework for Multirobot Systems. In: Iocchi, L., Matsubara, H., Weitzenfeld, A., Zhou, C. (eds.) *RoboCup 2008. LNCS*, vol. 5399, pp. 568–579. Springer, Heidelberg (2009)