Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007)

Solving Multi-agent Control Problems Using Particle Swarm Optimization

Maciej A. Mazurowski, *Student Member, IEEE* and Jacek M. Zurada, *Fellow, IEEE*

Computational Intelligence Laboratory
Electrical and Computer Engineering Department
University of Louisville
Louisville, KY 40292
{mamazu01, jmzura02}@louisville.edu

***Abstract*— This paper outlines an approximate algorithm for finding an optimal decentralized control in multi-agent systems. Decentralized Partially Observable Markov Decision Processes and their extension to infinite state, observation and action spaces are utilized as a theoretical framework. In the presented algorithm, policies of each agent are represented by a feedforward neural network. Then, a search is performed in a joint weight space of all networks. Particle Swarm Optimization is applied as a search algorithm. Experimental results are provided showing that the algorithm finds good solutions for the classical Tiger Problem extended to multi-agent systems, as well as for a multi-agent navigation task involving large state and action spaces.**

I. Introduction

Many real world problems can be modeled as multi-agent control problems. Known examples are air traffic control, network management, distributed vehicle monitoring and robot control [1]. In essence, a problem, where decisions have to be made about actions of more than one entity in the system can be described in a multi-agent systems framework. Decision making in multi-agent systems has been gaining increasing attention over the past years [1]. A variety of theoretical frameworks have been proposed to formulate such problems [2], [3], [4], [5], and various methods were examined to solve them [2], [6], [7], [8].

This article addresses the finding of optimal acting policies for agents given a specific task. A simple centralized solution for these problems is to introduce one decision-making entity which acts according to its knowledge about the system state. When a system state is fully observable, this problem can be formalized as the problem of finding an optimal solution for Multi-agent Markov Decision Process (MMDP) [9]. MMDP is a simple extension of Markov Decision Process, where an action is replaced by a joint action of all agents in a system. Standard single-agent methods such as Dynamic Programming can be applied to solve these problems.

Centralized control as presented above, however, has multiple drawbacks. First, the entire system relies on one decision-making entity, which limits reliability of the system. Second, the information has to be exchanged between the decision-making entity and the entities which perform actions, which usually introduces an additional cost and again reduces system reliability.

To overcome these drawbacks, decentralized models were developed. In these models, decisions about actions are made by each agent individually according to its own observations. The observations are usually limited and do not cover a whole system state. A planning stage, however, can be performed in both the centralized and decentralized manner (one learner and multiple learners, respectively). One such model, called Decentralized Partially Observable Decision Process (Dec-POMDP) [3], [4], and its extension to infinite observations and actions spaces will be described in details and used as a theoretical framework in the following parts of this article.

To solve the problem, Particle Swarm Optimization (PSO) [10] will be used. Some applications of PSO to multi-agent systems have been reported in the scientific literature. Usually the same principles as those used in PSO are used to optimize collective behavior of agents in the system (e.g. [11]). Another example applications of PSO that are related to the research presented in this article are strategy search for the Iterated Prisoners Dilemma [12] and policy search for single-agent Markov Decision Processes [13].

Here, PSO algorithm is applied to sets of feedforward neural networks in order to find approximate solutions for decentralized multi-agent control problems. PSO has been used multiple times to optimize weights of a neural network [14], [15], [16], [17] with satisfactory results. Here, it will be applied to train multiple feedforward neural networks at a time, where each network represents the policy of one agent.

The proposed algorithm is an approximate one. It means that it is not guaranteed to find an optimal solution (neither local nor global). The need for approximate algorithms arises, however, due to the fact that any algorithm guaranteed to find an optimal solution is computationally prohibitive even for moderately large state and action spaces [3], [4].

The article is organized as follows. The first section covers an introduction to Particle Swarm Optimization. The second section offers more detailed description of Dec-POMDP. The third section demonstrates how PSO can be applied to solve Dec-POMDP. The final sections show the application of the

1-4244-0708-7/07/$20.00 ©2007 IEEE

proposed approach to two problems: a multi-agent Tiger Problem and a navigation on the plane problem.

II. Particle Swarm Optimization

Particle Swarm Optimization (PSO), introduced in 1995 by Kennedy and Eberhart [17], is an optimization tool inspired by the observation of bird flocking and fish schools, where interesting collective behavior emerges on simple individual policies. In PSO, a domain space is searched by a set of particles. Each particle is characterized by its position $\vec{x_i}$ and velocity $\vec{v_i}$. Fitness of each particle can be evaluated as $f(\vec{x_i})$, where f is a fitness function that is being optimized. For each particle, its best position $\vec{p_i}$ is stored. Each particle also has access to the best position reached by particles in its neighborhood, $\vec{p_g}$.

The algorithm proceeds as follows. First, positions and velocities of all particles are set randomly in a certain range. Then, in each iteration the velocity components of particles are modified according to the following formula [10] [18]:

$$v_{id} = wv_{id} + \varphi_{i1}(p_{id} - x_{id}) + \varphi_{i2}(p_{gd} - x_{id})$$

where φ_1 and φ_2 are random numbers between 0 and c_1, and 0 and c_2, respectively and w is an inertia coefficient. Thus, in general, velocity is adjusted toward the locally best value and the globally best value in a partially random manner. The position of a particle at the end of each iteration is updated as

$$x_{id} = x_{id} + v_{id}$$

Both $\vec{x_i}$ and $\vec{v_i}$ can be arbitrarily bounded. The algorithm stops when certain criterion on $\vec{x_i}$ (usually on its fitness $f(\vec{x_i})$) is satisfied. The output of the algorithm is the best position found.

Even though Clerc and Kennedy in [10] proved convergence of the algorithm in special cases, PSO in general has not been proven to converge to a global optimum. It is, however, widely used because of its simplicity and often excellent results for various problems. In particular, it has been reported to provide good results in optimizing weights for neural networks [17].

III. Decentralized Partially Observable Markov Decision Processes

Here, PSO will be used to find optimal solutions for decentralized multi-agent control problems. As a formal framework, Decentralized Partially Observable Markov Decision Processes (Dec-POMDP) will be used. The proposed algorithm, however, can be applied to other problems such as the Partially Observable Stochastic Game [5]. Dec-POMDP was introduced in [3] and further elaborated in [4].

Decentralized Partially Observable Markov Decision Process for two agents (for simplicity) is a tuple

$$(S, A_1, A_2, P, R, \Omega_1, \Omega_2, O, T, K)$$

where

- S is a finite set of states
- A_1 and A_2 are finite sets of actions for agents α_1 and α_2
- $P : S \times A_1 \times A_2 \times S \rightarrow [0, 1]$ is a transition probability function, which assigns a probability for a transition from each step to another given actions taken by agents
- $R : S \times A_1 \times A_2 \times S \rightarrow \mathbf{R}$ is a reward function. $R(s, a_1, a_2, s')$ is a reward provided for the system, when actions a_1 and a_2 are taken and the system changes its state from s into s'.
- Ω_1, Ω_2 are sets of possible observations of agents
- $O : S \times A_1 \times A_2 \times S \times \Omega_1 \times \Omega_2 \rightarrow [0, 1]$ is an observation probability function. $O(s, a_1, a_2, s', o_1, o_2)$ is a probability of observations o_1 and o_2 of agents α_1 and α_2, respectively, when actions a_1 and a_2 are taken in a state s and system changes its state into s'.
- T is a horizon.
- K is a threshold value.

The solution for the problem is a joint policy

$$\pi = (\pi_1, \pi_2)$$

where

$$\pi_i : \Omega_i \rightarrow A_i \tag{1}$$

for $i \in \{1, 2\}$. π_i are called local policies, because they map sequences of local observations into actions of an agent.

In centralized control, actions can be taken based on the system state or the set of observations of all agents. Here each agent, when deciding on an action, has access only to its own partial view at the world.

Even though the solution for this problem is a tuple of local policies, the solution itself can be found in a centralized manner. It has been shown that algorithms finding an exact solution for this problem are intractable even for moderate amounts of agents, states and actions (problem is NEXP complete in the worst case [5], [4]). Thus, there arises a need for the approximate solutions. The authors suggest using Particle Swarm Optimization to find such solutions.

In this article, in addition to the Dec-POMDP presented above, the authors will consider its extension (which makes it harder to solve, but more general), where sets S, Ω_i and A_i can be infinite for all i.

It can be noted that a method of solving Dec-POMDP can be applied to many other problems that can be treated as special cases of Dec-POMDP. For example, when the system state is fully observable for all agents, Dec-POMDP can be simplified to Dec-MDP. When there is only one agent in the system, Dec-POMDP becomes POMDP. And finally, when both conditions are satisfied, we deal with the simple MDP. The proposed algorithm can be applied to all these variations.

IV. Policy search and representation of joint policies

One of the approaches to solving multi-agent problems is a direct policy search. In such approach the joint policies are

evaluated one by one, where the choice of policies to evaluate is a matter of a particular algorithm. Solving a problem in this way is clearly centralized, i.e. behaviors of agents are being developed all at a time by a single learning algorithm. Because of the form of policies (see (1)), however, control stays decentralized.

The simplest approach to a direct policy search is to evaluate all possible joint policies and choose the one with the best value (the brute force approach). It is, however, usually intractable to search through all the policies, even for a moderate amount of states, actions and agents. A simple example can be given. A number of joint policies for the problem is equal to $N_A^{N_S I}$ where N_A is a number of possible actions, N_S is a number of possible observations of each agent (for this example it is assumed that agents are homogenous) and I is a number of agents. It can be easily calculated that for the simple task of navigation in a 10×10 grid-world with 4 possible actions of each of two agents the number of joint policies reaches 10^{120}. Evaluating this amount of joint policies is clearly computationally prohibitive. It follows that finding an optimal joint policy in this way is impossible when amount of states or actions is infinite, as there is infinite amount of joint policies for these cases.

Nair et al. in [6] suggest Joint Equilibrium-Based Search for Policies. In this algorithm policies of all agents, except for one, are fixed and the best policy for the unfixed agent is found either by an extensive search or by using Dynamic Programming. This method is guaranteed to find a local minimum although there is still a large computational expense (which implies intractability for large state and action spaces).

Here, the authors suggest using Particle Swarm Optimization to search for the best solution in a joint policies space. In order to do so, joint policies must be represented in terms of position of the particle. Thus, the following function must be constructed:

$$m : \vec{x} \rightarrow \pi$$

The question arises as to why an entire joint policy is represented in a search space and why not represent one local policy at a time and optimize it. The reason is that that many multi-agent tasks require cooperation. This need for cooperation, in turn, implies that the overall optimal local policy for each agent may not exist and each policy can be evaluated only in the context of policies of other agents (unless special conditions on Dec-POMDP are satisfied). Thus, one can not find optimal local policies individually.

A simple example can be given. Two agents are approaching each other and their task is to pass each other without collision. They have two possible actions: to go on the right side of the road or to go on the left side. Neither action is better than the other. A choice becomes preferable only when the action of the other agent is known. Representing all the policies in one space allows for the fitness function to reflect a value of all local policies together (the authors however do not claim that this is the only appropriate approach to the problem).

The task of finding a function m seems to be crucial for the problem. The representation should satisfy some conditions in order to increase a chance for PSO to find a good solutions. First, the dimensionality of $\vec{x}$ should not be too large. Second, similar $\vec{x}$ should correspond to similar joint policies. The second heuristic allows the structure of an optimized function in a position space not to be completely random and may make the search easier.

One simple approach to construct a function m is to choose a dimensionality of $\vec{x}$ equal to the number of agents and to represent the policy of each agent in terms of the value of one component of $\vec{x}$. This mapping, however, has major drawbacks. The amount of policies for an agent is usually very large, which means dividing a finite range into a large number of parts ($N_A^{N_S}$), which results in tiny areas corresponding to one joint policy. Also, any representation of policy on the range will (at least occasionally) assign similar values of a component of $\vec{x}$ to very different policies. These two characteristics of analyzed mapping will cause significant difficulties in finding a good solution by PSO.

Another simple approach is to assign one component of $\vec{x}$ to each state of each agent and to represent actions performed in each state in terms of values of components. This will yield $N_S I$ components. A range of each component will have to be divided into N_A brackets, each corresponding to one action. This method is free of both drawbacks of the previous approach. It results, however, in high dimensionality of search spaces and does not allow for infinite state spaces.

The approach suggested in this article makes use of neural networks as parametric approximators of policies. This representation has been presented in the literature on both single-agent and multi-agent learning. The policy of each agent can be represented using a feedforward neural network in the following way: state sequences correspond to inputs of a network and actions are determined on the basis of the output of the network. Exact relations between observation sequences and inputs and between actions and outputs can be chosen depending on the problem. Some suggestions are given below.

The number of inputs can be simply equal to the maximum length of a state sequence. If the current state sequence available to an agent is of the length n and the maximum length is m, 0 (or any other number which does not represent any state) can be provided as all inputs from $n+1$ to m. When a neural network is used, a more natural representation of a state can be obtained. For example, an agent's coordinates on the plane can be provided as two inputs of a network. In the case of a sequence, two inputs would be needed for each element of the sequence.

There is also a large freedom of choice, in choosing correspondence between actions and outputs of a neural network. It can be done by assigning one action to each output and choosing the output with the greatest value. One can also use only one output and the action can be determined

on the basis of its value (this option allows for an infinite amount of actions).

A neural network (its weights, architecture and assumption about representation of states and actions) uniquely determines a policy. When a neural network for each agent is found, a joint policy is known. When the architecture of the NN is known, each network can be represented as a vector of weights. To represent a tuple of neural networks, the set of weights of all the networks must be used. This vector becomes an $\vec{x}$ for the PSO and thus a search is performed in weights space for for all the networks. The dimensionality of a resulting search space is $\sum_{i=1}^{I} N_{wi}$ where N_{wi} is a dimension of $\vec{w}_i$, i.e. a vector of weights of an agent's α_i network.

In the PSO algorithm, a reward function is used to construct a fitness function. For each joint policy π a fitness function is defined as $f(m^{-1}(\pi)) = \sum_{t=1}^{T} E(r^{\pi,t})$ where $r^{\pi,t}$ is a reward received in a time t, when a joint policy π is followed. The expected value of $r^{\pi,t}$ depends on functions R, P and O. Thus, fitness simply describes an average reward received when a joint policy is followed. In the optimization process the authors use a generic version of PSO presented in Section II.

V. Experimental results

In the experimental section two problems are presented. The first problem is the classical Tiger Problem expanded to multi-agent systems. It is shown that, even though the problem presents difficulties for the algorithm proposed in this article, the algorithm finds good solutions. The second problem is a problem of coordinated navigation, where two agents have to meet on a plane with obstacles. It will be discussed how properties of this problem make the proposed algorithm very suitable to solve it, and it will be shown that, in fact, good solutions were found.

A. Tiger problem

The tiger problem can be considered as classical in research on single-agent learning in partially observable environments. It has been extended to multi-agent systems by Nair et al. in [6].

In the multi-agent version of the problem two agents are standing together in front of two doors. Behind one door there is a tiger and behind the other door there are treasures. Thus there are two states of the system: SL and SR. Each of the agents in each turn receives one of two local observations and may perform one of three actions: open either door or listen. More formally:

- $\Omega_1 = \Omega_2 = \{HL, HR\}$
- $A_1 = A_2 = \{OpenLeft, OpenRight, Listen\}$

If both agents listen, then in the next turn they receive observation corresponding to the state of the system (this is a minor simplification of the problem, when sequences of only two observations are taken into account; in the classical formulation, there is uncertainty in acquiring information about the system state). Thus, if the state is SL, they receive the observation HL and similarly for SR. If they did not listen in the previous turn, they receive observations HL and HR with probabilities 0.5.

If either agent opens a door to the room with the tiger, they both receive a large penalty (smaller, when both do it at the same time). When either of them opens the door to a treasure, while the other door is closed, they receive a reward (greater, when both open the door at the same time). There is a small cost when they both listen.

The problem is to find an optimal set of local policies for the agents, thus one that guarantees the greatest expected reward. This problem is a finite horizon problem and can be formulated for any amount of turns. In this experiment, two turns will be taken into account. For more details on the problem see [6]

It must be noted that this problem has characteristics which make it difficult for the presented algorithm. First, the fitness function is not continuous in the weight space. Because of the finite number of policies and continuous weight space, certain subspaces of the weight space correspond to the same policy, and thus to this same fitness value. Such "flat" regions may make a search harder. Second, similar joint policies in this problem may correspond to a very different average reward and thus very different fitness. Since similar policies correspond to similar vectors in a search space, the resulting fitness function may contain numerous very steep and narrow hills and valleys, which again make the search harder. The purpose of this experiment is, however, to show that even for such task, good solutions can be found frequently.

To find a solution, two neural networks with the same structures were used, one network for each agent. Each neural network had two inputs 10 hidden neurons and three output neurons. Each input corresponded to one observation in a sequence. If a sequence consisted of only one observation, a value of 0 was provided to the second input. There was one action assigned to each output. The action with the largest corresponding output value was chosen.

As described in the theoretical section, all weights of all neural networks (2 in this case) serve as a domain space for the PSO algorithm.

The following test was performed: the algorithm was executed 100 times for a limited amount of iterations and the final result (expected reward) was noted. Then a random policy search was executed 100 times with the same amount of evaluated policies. Fig. 1 presents the distributions of the results of both PSO and random methods.

It can be seen that the proposed algorithm is capable of finding an optimal joint policy, which in this case corresponds to the expected reward of 144. This joint policy was found however in only 8% of cases. It is also visible that suboptimal policies corresponding to expected rewards of 100 and 122 are found in about 60% of cases, which shows that the algorithm can provide with large probability a good suboptimal solution for this problem. In addition, one can see that the solutions are not found by accident and the use of PSO in the policy search process is justified.

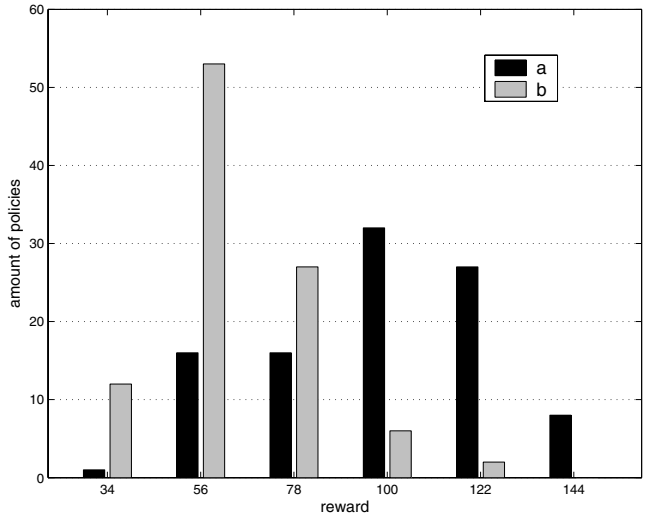

Fig. 1. Distribution of the results of a) the proposed algorithm and b) random policy search

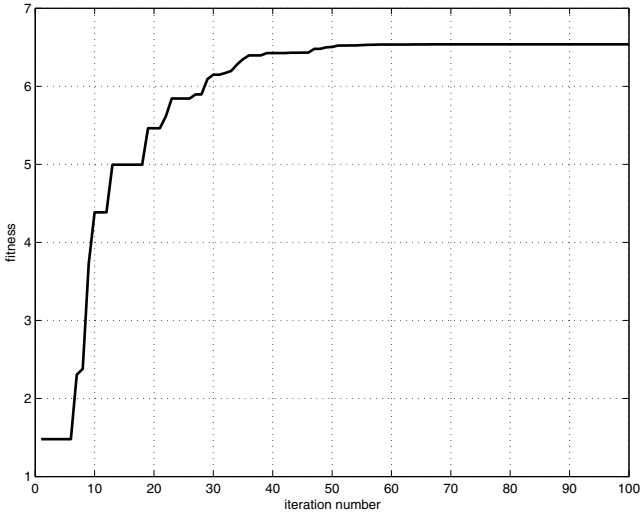

Fig. 2. Learning curve for the navigation task

Comparison of the results shows that the proposed algorithm offers significantly better results than a random policy search.

B. Navigation task

The second problem analyzed is one of coordinated navigation on a plane. Two agents (robots) start at position (5, 0.2) for Agent 1 and position (5, 7.5) for Agent 2. Their task is to get as close to each other as possible. There are walls between agents as shown in 3. Agent 1 can move a distance of 0.1 and agent 2 can move by the distance of 0.12 in each turn. Both agents can move in any direction and can observe their own positions. A more formal description of observation and action spaces follows:

- $\Omega_1 = \Omega_2 = [0, 10] \times [0, 10]$
- $A_1 = \{a_{\beta,0.1} : \beta \in [0, 2\pi]\}$
- $A_2 = \{a_{\beta,0.12} : \beta \in [0, 2\pi]\}$

where $a_{\beta,z}$ is the action of moving on the angle β by the distance z. It is clear that in this case the amount of possible observations and amount of possible actions for each agent are infinite. This fact makes it impossible to perform an exhaustive policy search, because of the infinite amount of resulting policies for each agent and thus infinite amount of joint policies. The JESP method presented in [6] is clearly inapplicable as well.

In general, any algorithm guaranteed to find an optimal solution is intractable in this case. It is enough to treat the spaces Ω_i and A_i as finite but arbitrarily large and apply the results from [3] and [4] to obtain this conclusion.

Here, one heuristic will be applied (all the comments above hold, even when this heuristic is applied). The search will be limited to memoryless policies also called reactive policies. Memoryless policies are simply functions that map current observations into actions, thus $\pi : \Omega \rightarrow A$. This heuristics is based on the assumption that for this task the information about the previous observations is not necessary to obtain an optimal behavior. This search limitation can be applied for each problem. In many tasks, however, memoryless policies provide poor results. On the other side, however, such limitation make the problem of finding policies easier. Nevertheless, it stays intractable for exact algorithms (at least NP-Hard) and JESP. It will be shown that the algorithm presented here will find good solutions for the problem.

The same neural network structure was used for both agents. Each network had two inputs, each corresponding to one coefficient of a position of the agent on the plane, and one output. The value of the output determined the angle of movement for the agent. There were 10 neurons in the hidden layer of each neural network.

The following parameters have been used for the PSO algorithm: $w = 0.4$, $c_1 = 0.8$, $c_2 = 0.6$. Iterations limit was set to 100 and population size was 2000.

Fig. 2 presents a learning curve for this task (in a very successful run of the algorithm). A reward here is a change of distance between agents (the larger the change, the better the joint policy). It can be seen that the joint policy with value about 6.5 is found after 50 iterations. Fig. 3 presents the paths of the agents when the best joint policy at the moment is followed (local policies are followed by agents individually) in four moments in the learning process. Significant improvement can be observed in each pictured iteration.

To conclude it can be said that the algorithm is capable of finding a very good solution for this problem. It also consistently produces good results in each run (an average reward of 5.68 with a standard deviation of 0.45 over 100 trials).

VI. Conclusions and further research directions

In this article the algorithm was proposed to find approximate solutions for decentralized multi-agent control problems. Decentralized Partially Observable Markov Decision

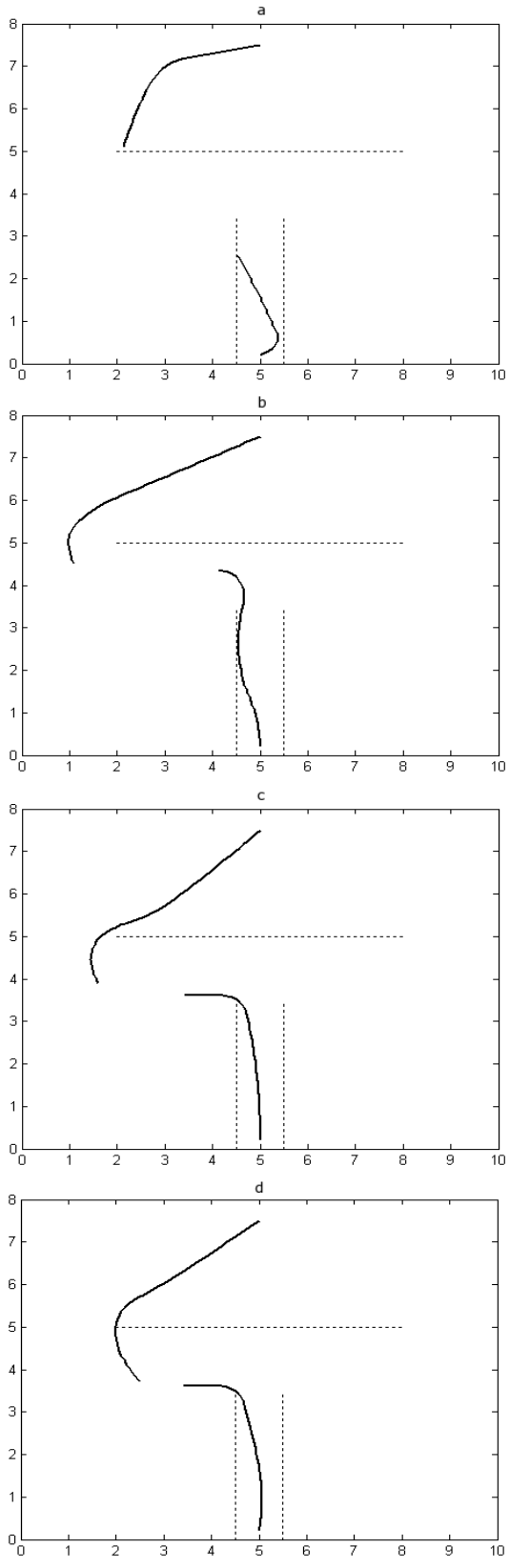

Fig. 3. Paths followed by agents when the best joint policy is used after a) random initialization of particles b) 10 iterations, c) 20 iterations d) 60 iterations. Dotted lines represent obstacles.

Processes were employed as a theoretical framework. PSO was used to optimize the weights of a collection of neural networks simultaneously, with each network representing one policy, and thus to find an optimal joint policy. The experimental results show that the algorithm provides good solutions for the Tiger Problem extended to multi-agent systems. Real advantages over other algorithms, however, become apparent when a problem with infinite (or very large) state and action spaces is considered. The multi-agent navigation task was used as an example of such a problem to show the efficiency of the proposed algorithm.

Future research will cover more detailed theoretical consideration on applicability of the algorithm and tests on real-world problems.

ACKNOWLEDGEMENTS

The authors would like to thank Piotr A. Habas and Dr. Mehmet K. Muezzinoglu for their helpful comments and inspiration and Katie Todd for her help in preparation of this article. The authors would also like to thank three anonymous reviewers for their valuable comments.

REFERENCES

[1] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, 2005.

[2] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proceeding of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, July 1998, pp. 746–752.

[3] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of Operations Research*, vol. 27, pp. 819–840, 2002.

[4] C. V. Goldman and S. Zilberstein, "Decentralized control of cooperative systems: Categorization and complexity analysis," *Journal of Artificial Intelligence research*, pp. 143–174, 2004.

[5] E. A. Hansen, D. S. Bernstein, and S. Zilberstein, "Dynamic programming for partially observable stochastic games," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004.

[6] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella, "Taming decentralized pomdps: Towards efficient policy computation for multiagent settings," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003, pp. 705–711.

[7] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling, "Learning to cooperate via policy search," in *Sixteenth Conference on Uncertainty in Artificial Intelligence*, San Francisco, 2000, pp. 307–314.

[8] S. Seuken and S. Zilberstein, "Formal models and algorithms for decentralized control of multiple agents," University of Masschusetts, Amherst, Computer Science Department, Tech. Rep., 2005.

[9] C. Boutilier, "Sequential optimality and coordination in multiagent systems," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, July/August 1999, pp. 478–485.

[10] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58–73, February 2002.

[11] F. Chiang and R. Braun, "A nature inspired multi-agent framework for autonomic service management in ubiquitous computing environments," in *Computational Intelligence Methods and Applications*, 2005.

[12] N. Franken and A. P. Engelbrecht, "Particle swarm optimization approaches to coevolve strategies for the iterated prisoner's dilemma," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 562–579, December 2005.

[13] H. S. Chang, "An adaptation of particle swarm optimization for markov decision processes," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, 2004, pp. 1643–1648.

[14] L. Messerschmidt and A. P. Engelbrecht, "Learning to play games using a pso-based competitive learning approach," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 280–288, June 2004.

[15] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural network," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, SIS '03*, 24-26 April 2003, pp. 110–117.

[16] R. Mendes, P. Cortez, M. Rocha, and J. Neves, "Particle swarms for feedforward neural network training," in *Proceedings of the 2002 International Joint Conference on Neural Networks, IJCNN '02*, 2002, pp. 1895–1899.

[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceeding of IEEE International Conference on Neural Networks*, Piscataway, NJ, 1995, pp. 1942–1948.

[18] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, June 2004.