The CSI Journal on Computer Science & Engineering

The CSI Journal on
Computer Science and Engineering
Vol. 1, No. 2(b), Summer 2003
Pages 39 - 44
Short Paper

Using Layered GA in Multi-agent Learning

Ali Nouri Jafar Habibi

Department of Computer Engineering
Sharif University of Technology, Tehran, Iran

Abstract

In order to successfully apply evolutionary learning to the increasing complex Multiagent Systems, we must develop new techniques to improve the performance of the learning process. Evolutionary algorithms, by themselves are very slow; when applied to a complex multiagent system, convergence problem becomes more critical since appropriate coordination strategies between the constituent parts should be achieved and this highly increases the complexity of the problem [1]. Here, we successfully apply a layered Genetic Algorithm to evolve behavioral strategies in a multiagent environment namely *The Pursuit* problem. Experimental results show that this method outperforms single layer GA and an auction based traditional algorithm.

Keywords: Evolutionary learning, multi-agent systems, convergence problem, genetic algorithms, layered genetic algorithm

1. Introduction

Multiagent systems propose new challenges toward using evolutionary learning. When agents evolve to perform a task in a MAS, they should learn not only to concern with their own goals, but also to be able to adapt themselves to their environment including other agents. This adaptation may be formed in the sense of cooperation or competition with other agents [12,15,16]. In this case, evolved behavior in combination with other agents' strategies makes up the entire solution. This is in contrast with evolution in single agent environments where each individual in the population is typically considered as a complete solution to the problem.

When coevolution is used to evolve the desired behavior, complexity of the system state space grows drastically since several agents participate in the evolution simultaneously and hence simple genetic algorithms usually fail to achieve the expected behavior. This is also the case when agents have noisy sensors/actuators and are supposed to perform complex behaviors [21]. From this standpoint, several techniques have been used to overcome this problem. Gomez and Miikkulainen used incremental evolution in order to emerge complex behaviors. They started evolution with simple goals and gradually increased the difficulty of the expected behavior as the agents learn to perform simpler tasks [20]. Whiteson and Stone showed in a series of papers that layered learning can greatly improve the performance when coevolution is used in complex environments [17,18,19]. Here we successfully apply two-layered genetic algorithm to evolve behavioral strategies of predators in the Pursuit environment. This layering is different from previous approaches in that we use a separate population within each individual in order to perform a fine-tuning search, while in previous approaches, layering is based on the task itself rather than the state space.

Layered learning in here consists of an inner agent and an inter-agent learning; both layers use GA for evolution and a link is provided to connect these two layers. Experimental results show that this architecture outperforms both single layer GAs and auction based traditional algorithms.

The rest of this paper is organized as follows: after the introduction, in section two the predator-prey pursuit environment is introduced and its specifications are briefly described. Section three discusses some issues for applying GA to multiagent systems. We then describe the proposed method used in the Pursuit problem in section four and finally section five concludes the paper.

2. The Pursuit Problem

The Predator-Prey Pursuit game was first introduced by Benda et. al [2] and comprised 4 predators which try to capture a prey by surrounding it in 4 directions in a grid-world. So far, the game has been one of the finest environments to study multiagent systems and their properties. Simplicity and extendibility of the environment along with the potential power of agents to show complex behaviors in the form of simple actions have all caused the game to be a suitable environment to test multiagent strategies. By now, many different configurations have been suggested for the system [8,9,24]. Each set of these configurations leads to a specific level of complexity in the game. It is eligible to mention that changing these parameters will significantly change the performance of the agents and therefore it should be considered when comparing different strategies.

In the first version, each agent can only move into one of the four neighborhood cells around him. The prey moves randomly and stands still 10% of the cycles in order to simulate a slower movement. The world is surrounded by walls, meaning that prey can be captured using less than 4 predators by forcing the prey to move to the corners of the world. Agents can sense up to four cells away in each direction. Predators can see each other independent of their positions in the world and predators decide without using communication. Below is a list of important parameters of the system.

- **Definition of the capture scenario.** Different situations could be imagined as the capture scenario. While in some configurations, surrounding the prey is the only way to capture it, in other configurations, simultaneous movement of the prey and a predator to a same cell will cause the prey to be captured.
- **World.** Removing the walls from the environment greatly increases the complexity of the game since the prey can flee by rounding the world.
- **Action policy.** Two possible strategies could be considered here; serial movement lets the agents move in turn while in parallel movement, all agents move simultaneously. By choosing the agents to move simultaneously we are adding the task of agent-modeling to the agents goals and thus increasing the complexity further.

In our configuration, our aim is to make the task of predators as complicated as possible thus we have made the world without borders and let the agents to move simultaneously. The prey, in the other hand, moves as fast as the predators and uses a semi random strategy.

This set of parameters chosen for the system has greatly reduced the chance of capture by chance in the system. Along with these parameters, a new object is added to the environment namely *the obstacle*. Each obstacle occupies one cell and prevents other agents to move to that cell. This is to let predators evolve strategies which use obstacles to capture the prey.

3. Genetic Algorithms and Multiagent Systems

Applying Genetic Algorithms to a multiagent system requires a great care to be taken. GAs are by themselves slow solutions; applying them to a MAS drastically increases the complexity of the process and therefore convergence problem becomes more challenging. In a multiagent system using Division or Interaction mechanisms where each agent is supposed to be different from the others in terms of behaviors, beliefs or even goals [10], traditional GAs usually fail to emerge these characteristics since individuals are typically evaluated in isolation and represent the complete solution to the problem while in a multiagent system, complete solution comprises all of the individuals in the current generation [4].

In order to overcome the convergence problem, several techniques are used by different researches; one idea is to reduce the complexity of the system by letting each agent to perform separately [3]. This is done by evaluating each individual in a separate simulation. In each run, several copies of an agent are placed in the environment to perform the task. The main problem with this approach is that we're narrowing the search space down to a very small region where agents are equal in their behavioral strategies. Another problem is that in some cases the expected behavior is not met because individuals can not self trick. In other words, in systems where agents are supposed to compete with each other as well as to cooperate, like the Pursuit problem, agents typically fail to compete with themselves and therefore they either cause too many conflicts with each other resulting in a less opportunity of capture or they gradually lose their desire to compete, turning to a more cooperative strategy (collusion emerges instead of competition); This will cause them to fail when they face new opponents such as human experts [6].

Coevolution is also used to ameliorate this problem. Coevolution is beneficial in that it allows several agents to evolve at the same time. As the agents are performing simultaneously in the same environment, they get the chance of adapting themselves to other agents and the environment more actively and thus preparing to challenge new situations more easily [7]. This is specially the case when developing games that may be used later to challenge human experts. The problem with coevolution is the convergence problem; in fact building a coevolutionary system that practically converges is really hard when the individuals compete each other rather than just performing like parasites. The problem occurs when agents happen to follow each other rather than to learn new behaviors, i.e. Agents are not getting wiser but just regulating their actions to fit other agents. Another problem with traditional coevolutionary systems is that they're critically dependant on the nature of the environment which the learners are placed and the specific attributes of the task domain [11].

Sometimes new extensions are built in order to further ease the task of learning. In [12] representatives from other species are used to evaluate each agent in the population and therefore different species do not learn simultaneously in the same environment. This enhances the convergence process. Concurrent layered learning, Incremental learning and learning through task decomposition are also some other techniques, used to overcome the same problem [17,18,19,20].

Here we successfully apply a two-layered genetic algorithm to the population of predators in the pursuit problem. Predators perform simultaneously in the same environment

and are evaluated according to other agents. In the next section we describe the proposed algorithm in detail.

4. The Proposed Method

4.1 Constructing the environment

A grid world of size 20×20 is used as the environment. There is only one prey in the scene and its strategy is a semi-random one. It first finds the empty cells that can be moved into and then randomly selects one of them. The complete random method is not used because we want to have all the agents move at the same speed; if the prey moves in a complete random manner, it may be stopped by the conflicts with other predators resulting in a slower movement. On the other hand, a linear prey manages to escape the predators by moving in only one direction and causing all the predators to remain behind it. This problem is mainly due to the borderless attribute of the environment.

The predators use Case Base Reasoning [13] for their decision making. Figure 1(B) shows a typical Case of a predator. While each predator is considered as an individual in the population of agents, it serves as an environment which contains Cases or Rules (Figure 1(A)) within him. Each simulation run is 200 cycle and agents are positioned randomly in the environment at the beginning of each run.

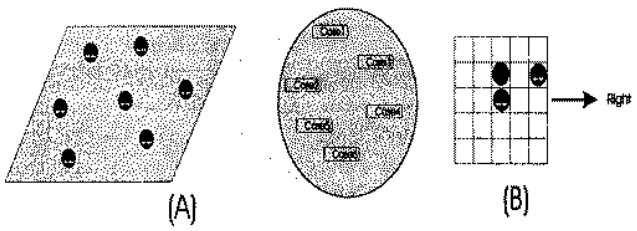

Figure 1. (a) Structure of populations in the system, (b) A typical Case.

4.2 Inner agent evolution

The genetic algorithm used inside agents is to equip each individual with learning ability during his lifetime. This GA performs evolution to the cases of the agent and starts when the predator is born and continues until he is dead. Below are the properties of the algorithm:

Individuals. As suggested above, individuals in this algorithm are the cases which the agent uses in his CBR. Each case is in the form of $situation \rightarrow action$ where situation is the vector of sensory information gained by the agent and Action is the corresponding movement to be taken in the range of (N,W,S,E).

Producing the first generation. First generation is built when the agent is born and it is automatically inherited from predator's parents. In fact, there is no need to make the start up population since the outer layer of evolution makes it. First generations of predators which are fed into the system at the beginning do not have any cases. They gradually add cases to their repository. Other strategies could also be imagined here; preliminary cases can be constructed using other methods such as neural networks, human experts, etc.

Executing Cases. Selecting a case to be executed in each cycle is done via a simple resemblance detection algorithm. Each case is checked against current situation to find the nearest one. A threshold is then used to check whether the case should be applied or not. If the distance between the two situations is too long then a new case is created with current situation and a random direction as the movement, and then it's added to the case repository. The threshold is a linear function of fitness of the selected case. The higher the fitness is, the farther it can be from the current situation. Below is the actual function used in the system.

$$Tresh(c) = 0.2\, fitness(c) + 3.0 \qquad (1)$$

Fitness Function. Fitness of each case is the sum of all the rewards it has received from the environment upon its execution divided by the number of times it is used so far. Every time a case is executed, its reward is calculated by the environment from the point that how beneficial the movement has been, i.e. a reward is the difference between the value of the situation in current cycle and the value after execution of the case. Value of a situation is calculated as follows:

1. Add 2.5 units if the agent itself is in the neighborhood of the prey.
2. Add 1 unit if another predator is in the neighborhood of the prey.
3. Add $\frac{1}{MD(prey, predator_i)}$ for $i=(1,2,...,number\ of\ predators)$ and MD is the Manhattan Distance \footnote{Sum of x and y offsets} between two agents.

Evolution Process. Evolution is applied every 3 cycles. In order to generate next population, tournament selection [14] is used. Single point crossover is used to construct the situation part of the offspring case. The action part is chosen from one of the parents with a 50% probability. Mutation operator only changes the action part to a new random direction and is applied with a variable rate starting from 5% at the beginning and slowing down as time goes by. The fitness of the newly generated case is set to a constant higher than zero to increase the durability of the cases which have not been executed yet (good cases may not have yet the opportunity to show their abilities).

Once a new offspring is born, we decide whether to insert the case into the repository or to replace another case. For this, the most similar case to this offspring in the population is found and then replaced by it if its fitness is lower; otherwise the offspring is simply inserted into the repository. We use a threshold for the number of cases to avoid explosion of population. If the number of cases reaches the threshold, a suicide attack is used to eliminate poor-valued cases.

4.3 Outer layer evolution

This layer of the algorithm is applied to the population of predators as individuals. The evolution is competitive as well as cooperative. It is cooperative since agents share the same goal and should cooperate with each other in order to capture the prey. Once the prey is captured, the reward is sent back to participating agents. And it's competitive because only predators surrounding the prey receive special rewards for capturing the prey. Agents who rarely participate in captures are eliminated from the population. Below are the properties of this layer:

Structure of Individuals. Predators are the participants of the evolution in this layer. Encoding is achieved by concatenating all the cases in the CBR repository.

Fitness Function. Fitness function consists of two parts which are used in previous researches [3]. The first part measures the participation level of the agent in prey capture. Every time the prey is captured, the surrounding predators receive a constant amount of energy *P*.

The next part is used to fine-tune the evaluation process. Since predators hardly capture the prey in the beginning of the simulation, this method is used to give more energy to predators who manage to maintain a minimum distance to the prey in their lifetime. This function adds $\frac{1}{\mathrm{MD}(\mathrm{self},\mathrm{prey})\times \mathrm{age}(\mathrm{self})}$ to the agent's energy in each cycle where *MD* is the Manhattan Distance and *Age* is the age of agent. Of course this amount is insignificant in comparison with the constant *P* and only serves as the fine tuning evaluation.

Evolution Process. Each generation lasts for 3 simulation runs, i.e. 600 cycles. After this period, tournament selection is used to find some parents from the population. Then a special kind of crossover operator is used. Parents' cases are either selected to be inherited by the offspring or to be thrown away according to the function *SP*. Here, we intend to reduce the number of cases by approximately $\frac{3}{4}$ and at the same time allow cases with higher fitness to have more chance to be inherited.

$$\mathrm{SP}(\mathrm{case}) = \frac{5\times \mathrm{fitness}(\mathrm{case})}{100} \qquad (2)$$

Unlike the inner layer, here we never allow the offspring to be added to the population; in return we always replace the weakest individual by the newly born predator. This is because each agent has had enough opportunity to perform and lower energy really means weaker strategy.

Linking two layers. The two layers are closely related to each other. While the outer layer finds the solution to the problem, the inner layer helps each individual to learn independently in its lifetime to further avoid collusion problem which usually happens in coevolutions. The outer layer provides the newly born predators with a startup population of cases and at the same time inner layer contributes to the energy of the agent and therefore affects his position in the population.

5. Experimental Results

Figure 2 shows the effectiveness of the algorithm in capturing the prey during the first 1000 generation (this adds up to 600,000 cycle of simulation). Figure 3 shows maximum, average and minimum energy of the predators in each generation. As it is shown in the chart, energy function is not changed a lot during the simulation. After a few generations, predators manage to stay around the prey but they lack useful coordination strategies to successfully capture the prey. Soon after, they find new coordination strategies and manage to capture the prey more often.

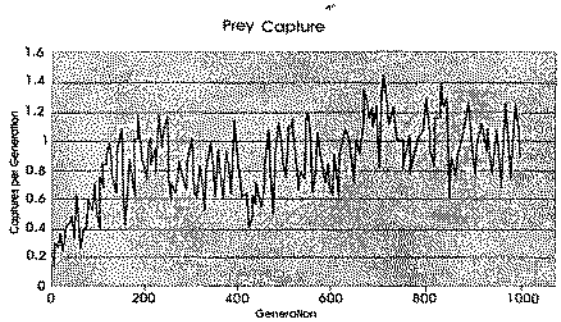

Figure 2. Average number of captures in the first 1000 generation

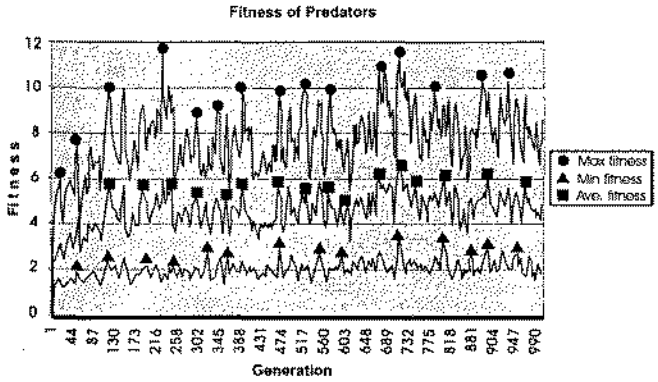

Figure 3. Maximum, Minimum and Average of predators' energy levels

In Table 1 the result of this algorithm is compared to a single layer GA and also an auction based traditional algorithm. The algorithm outperforms single layer GA both in the speed of convergence and the overall performance of the predators. Interestingly, the average energy of agents in single layer method is slightly higher than the energy function with two layered approach which shows that predators manage to stay around the prey better when using single layer approach but the conflicts between predators doesn't allow them to capture the prey effectively.

Strategies which cause the agent to stay close to the prey receive instant rewards from environment while strategies which consider agent coordination should wait until a capture scenario occurs. Single layer GA does not distinguish the benefit of these strategies and is more interested in selecting strategies that can receive instant reward and therefore is more susceptible to be trapped in local minimums. That's why the average energy function is higher in single GA approach.

6. Conclusions and Future Work

In this paper, we showed that a two-layer genetic algorithm can be used to evolve complex behavioral strategies in multiagent systems. The layering is based on the state space and causes a hierarchical searching paradigm. While the outer layer performs a coarse search, the inner layer fine-tunes the results by providing learning ability for predators in their lifetime. While the layering causes a wider search scheme, resulting in an appropriate amount of diversity in the patterns, evolution in the inner layer causes a boost toward efficient sub-structures.

Table 1. Result of Three different algorithms for predators

Results/Algorithm	Auction	Single layer GA	Multi-layer GA
Captures in each run	70%	73%	81%
Average Energy	4.04	4.94	4.87

Future works will include testing this approach in other environments. One of these suitable environments is the Keepaway Task [23] which is an extension to the Simulated RoboCup soccer environment. The environment is like the pursuit game in the way that multiagent coordination is highly required for the team to survive but the behaviors of the agents are more complicated. Another system to test the proposed solution is a new form of the classical pole balancing problem [25]. In the new form, several agents are put around the carriage containing the pole. Agents can only force the carriage forward and carriage can move omni directional. The goal of the whole team is to keep the balance of the pole. Another set of experiments could be performed with decision making methods other than CBR. ESP [22] and a fuzzified CBR are good candidates.

References

[1] J. Habibi, M. Ahmadi, A. Nouri, M. Sayyadian and M. Nevisi, "Implementing Heterogeneous Agents in Dynamic Environments, a Case Study in RoboCupRescue," *First German Conference* MATES-2003.

[2] M. Benda, V. Jagannathan and R. Dodhiawala, "On Optimal Cooperation of Knowledge Sources-an Empirical Investigation," Technical Report BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, July 1986.

[3] T. Haynes and S. Sen, "Evolving Behavioral Strategies in Predators and Prey," Gerhard Weib and Sandip Sen, editors, Adaptation and Learning in Multiagent Systems, Lecture Notes in Artificial Intelligence, *Springer Verlag*, Berlin, Spring 1996.

[4] A. Nouri, "Contrasting Multiagent and Single Agent Learning in a Single Environment,", B. S. Project Report, Computer Engineering Department, Sharif University of Technology, July 2003.

[5] D. H. Ackley and M. L. Littman. "Altruism in the Evolution of Communication," In Rodney A. Brooks and Pattie Maes, editors, Artificial Life IV: *Proc. of the Intl. Workshop on the Synthesis and Simulation of Living system, MIT Press*, November 1994.

[6] B. J. MacLennan and G. M. Burghardt, "Synthetic Ethology and the Evolution of Cooperative Communication," *Adaptive Behavior*, 2(2), pp.161-188, 1993.

[7] K. Jim, C. Lee Giles, "Talking Helps: Evolving Communicating Agents for the Predator-Prey Pursuit Problem," *Artificial Life*, 6(3), pp. 237-254, MIT Press, 2000.

[8] R. Levy and J. S. Rosenschein, "A game theoretic Approach to the Pursuit Problem," In Working Papers of the *11th International Workshop on Distributed Artificial Intelligence*, pp. 195-213, February 1992.

[9] L. M. Stephens and M. B. Merx, "The Effect of Agent Control Strategy on the Performance of a DAI Pursuit Problem," *In Proc. of the 1990 Distributed AI Workshop*, October 1990.

[10] G. Weib, "What is 'Multi' in Multi-agent Learning?" Weiss, G., Dillengourg P. (1999), In P. Dillenbourg (Ed) *Collaborative Learning: Cognitive.*

[11] A. D. Blai and J. B. Pollack, "What Makes a Good Co-evolutionary Learning Environment?" *Australian Journal of Intelligent Information Processing Systems 4*, 166-175, 1997.

[12] M. A. Potter and K. A. De Jong, "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents," *Evolutionary Computation*, MIT Press 8(1), pp.1-29,2000.

[13] J. L. Kolodner, *Case-Based Reasoning, Morgan Kaufmann Publishers* 1993.

[14] T. Blicke and L. Thiele, "A Mathematical Analysis of Tournament Selection," in L. Eshelman, editor, Proc. of the 6th Intl. Conf. on Genetic Algorithms (ICGA95), San Francisco, CA, Morgan Kaufmann publishers, 1995.

[15] C. D. Rosin and R. K. Belew, "Methods for Competitive Coevolution: Finding Opponents Worth Beating," in S. Forrest, editor, *Proceedings of the 6th Intl. conf. on Genetic Algorithms,* pp. 373-380, San Mateo, CA, Morgan Kaufman 1995.

[16] C. H. Yong and R. M., "Cooperative Coevolution of Multi-agent Systems", Technical Report AI01-287, The university of Texas at Austin, Department of Computer Sciences, 2001.

[17] P. Stone and M. Veloso, "Layered Learning", *Eleventh European Conference on Machine Learning* (ECML-2000).

[18] S. Whiteson and P. Stone, "Concurrent Layered Learning," *In 2nd Intl. Joint Conf. on Autonomous Agents and Multiagent Systems*, July 2003.

[19] S. Whiteson, N. Kohl, R. Miikkulainen and P. Stone, "Evolving Keepaway Soccer Players Through Task Decomposition,", *Proceedings of the Genetic and Evolution- ary Computation Conference, GECCO-2003.*

[20] F. Gomez and R. Miikkulainen, "Incremental Evolution of Complex General Behavior", Technical Report AI96-24S, June 1996.

[21] S. Luke, C. Hohn, J. Farris, G. Jackson and James Hendler, "Co-evolving Soccer Sofbot Team Coordination with Genetic Programming," In Hiroaki Kitano, editor, *RocoCup-97: Robot Soccer World Cup I,* Berlin, Springer Verlag, pp. 398-411, , 1998..

[22] F. Gomez and R. Miikkulainen, "Learning Robust Nonlinear Control with Neuro-evolution," Technical report AI01-292, The University of Texas at Austin Department of Computer Science , 2001.

[23] P. Stone and R. S. Sutton , "Keepaway Soccer: A Machine Learning Test bed," *In RocoCup-2001: Robot Soccer World Cup V.* Springer Verlag, Berlin 2002.

[24] G. Konidaris, D. Shell, N. Oren, "Evolving Neural Networks for the Capture Game," *In Proc. of the SAICSIT Postgraduate Symposium,* Port Elizabeth, South Africa, September 2002

[25] D. Michie and R. A. Chambers. "BOXES: An Experiment in Adaptive Control," In E. Dale and D. Michie, editors, *Machine Intelligence 2.* Oliver and Boyd, 1968.

Ali Nouri is a graduate of Alborz High School; he received his B.S. degree in Software Engineering from Sharif University of Technology in Tehran, Iran in 2003. His research interests are in the areas of Machine Learning, Multi Agent Systems, Evolutionary Learning and Artificial Neural Nets.
Email: nouri@ce.sharif.edu

Jafar Habibi received his B. S. degree in computer engineering from the Supreme School of Computer, his M. S. degree in industrial engineering from Tarbiat Modares University and his Ph.D. degree in Computer engineering from Manchester University. At present, he is a faculty member at the Computer Engineering department at Sharif University of Technology and the managing directorputing Mschine Services. He is also the president of the Computer Society of Iran and the supervisor of Sharif's Robo-Cup Simulation Group. His research interests are mainly in the areas of computer engineering, simulation systems, MIS, DSS and evaluation of computer systems performance.
Email: habibi@sharif.edu