Evolving Cooperative Groups: Preliminary Results

Maria Gordin, Sandip Sen, and Narendra Puppala
Department of Mathematical & Computer Sciences
The University of Tulsa
600 South College Avenue,
Tulsa, OK 74104-3189
e-mail:[gordin,sandip,puppala]@euler.mcs.utulsa.edu

Abstract

Multi-agent systems require coordination of sources with distinct expertise to perform complex tasks effectively. In this paper, we use a co-evolutionary approach of using genetic algorithms (GAs) to evolve multiple individuals who can effectively cooperate to solve a common problem. We concurrently run a GA for each individual in the group. We experiment with a room painting domain which requires cooperation of two agents. We have used two mechanisms for evaluating an individual in one population: (a) pair it randomly with members from the other population, (b) pair it with members of the other population in a shared memory containing the best pairs found so far. Both the approaches are successful in generating optimal behavior patterns. However, our preliminary results exhibit a slight edge for the shared memory approach.

Introduction

Our goal is to generate behavior strategies for cooperative agents that have distinct capabilities. A coordinated group effort is necessary to successfully complete the task. Performance of an agent depends on its performance in a group. Ability of agents to adapt to each other and cooperate becomes critical in this context. We have been investigating co-evolutionary methods to find such strategies in an agent group consisting of two agents. We use simultaneous evolution of two genetically distinct populations. Each population consists of rules (behavioral strategies) for one of the agents. The fitness of an individual in either of the populations depends on its ability to perform the task in a group with an individual from another population. We use a shared memory to store the n best couples found so far. The shared memory is also used to evaluate individuals from each of the co-evolving populations.

Simultaneous evolution approaches have been successfully used in both cooperative and competitive environments (Haynes & Sen 1997; Haynes *et al.* 1995; Ito & Yano 1995). Rosin and Belew (Rosin & Belew 1995) successfully used co-evolution for developing efficient strategies for two players games like Tic-Tac-Toe, Nim, and Go. Authors have used the SAMUEL system to co-evolve robots searching for food (Grefenstette & Daley 1996; Potter, Jong, & Grefenstette 1995). One of the approaches investigated in this work was to evolve strategies by using evaluation of individuals from one of the populations against random members of the set of previous champions from another population. Our approach of storing and using the best agent groups can be seen as a novel approach to co-evolution. Grefenstette's approach is more suited to competitive domains, whereas our approach is particularly tailored for cooperative problems.

Shared Memory in GAs

Though our shared memory approach can be used with larger groups, in this paper we use a domain with two cooperative agents. Both agents need to contribute to complete a shared task. Each of the agents has unique capabilities necessary to complete the job. We use a GA to find behavioral rules for each agent which will allow them to complete the task most efficiently. Performance of agents as a group depends not only on the ability of each agent to perform its share of the task, but also on the ability of the agents to adapt to each other and cooperate effectively. This means that evaluation of an agent's performance needs to be done in the context of a group.

In our initial experiments, we ran two different GA populations (one for each agent) concurrently. To evaluate an individual from the first population, we paired it with an individual from the second population. One way of doing it was to randomly pick an individual from the other population to construct a pair. However, in this approach, we did not have a way of remembering the successful pairs, as only the best individuals from each population were available, and the best individuals paired may not produce the best pair. We tried to overcome this problem with a shared memory

approach, the shared memory being a collection of the most successful pairs observed so far.

Another approach to evaluating individuals would be to pair it with one or more good individuals from the other population. Others have suggested using a random collection of the best individuals from the other population over all generations for this purpose (Grefenstette & Daley 1996). A variation of this approach would be to select a collection of best partners from the other generation. Rather than storing the best individuals separately, we chose to store the best pairs observed so far in a shared memory.

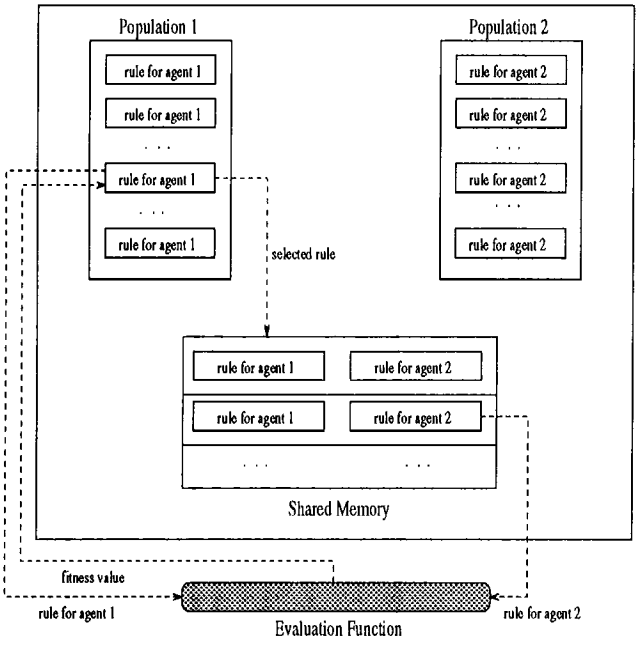

Figure 1: Using shared memory with two GA populations. The figure illustrates the process of evaluating an individual from Population 1.

If a pair performs better than any other pair, it could be because one of them performed really well, even if the other was not that effective. To improve the performance of such a pair, only one of the players needs to be replaced. It means that we need to pair the successful individual from one of the populations, with different individuals from another population, hoping to improve the result of the group. This can be achieved with a shared memory containing effective agent groups. In a competitive setting, the evaluation of an individual should be some kind of an average over its evaluation when pitted against a sampling of individuals from the other population (Grefenstette & Daley 1996; Rosin & Belew 1995; Haynes & Sen 1996). This is because we want to evolve robust individuals. In a cooperative setting, however the highest evaluation an individual receives from all pairings with others appear to be the best choice for the evaluation of that individual. This is because we are interested in finding a pair that performs the best. So, an individual need not be robust, it just has to effectively co-adapt with another individual.

Shared memory consists of a collection of pairs of individuals that have demonstrated the best performance so far (see Figure 1). Figure 1 also illustrates the process of evaluation of an individual from Population 1 by pairing it with an individual from Population 2 stored in shared memory. In our experiments, we evaluated an individual from population 1 by pairing it with each of the individuals from Population 2 stored in shared memory. The maximum evaluation from all such pairs is used as the evaluation of the individual from population 1. The shared memory gets updated if we encounter a fitness value higher than at least one of the fitness values of the pairs that have been stored in shared memory previously. In such a case, the pair with the minimum value gets replaced with the new pair.

Room Painting Domain

For evaluating our proposed cooperative co-evolution mechanism, we have defined a problem of room painting. A room with a boundary is divided into many empty squares which needs to be painted. Two agents are required to do the job: whitewasher and painter. A square can be painted only after it has been whitewashed. If the square is not painted within a particulat time interval after it has been whitewashed, the square turns back empty (three time ticks in our experiments). Both agents perform their tasks as they move. They can move in four directions: north, south, east and, west or, hold (stay where they are). The whitewasher moves twice as fast as the painter. To complete the job in a reasonable time, the painter needs to learn to follow the whitewasher and the whitewasher needs to learn to wait for the painter if it gets too far ahead.

We now explain the working of the simulation used for executing agent behaviors. We use the following alphabet to represent states of a square: 0=empty, 1=painted or whitewashed, 2=anything, 3=whitewashed, 4=empty or painted.

Direction to move	North	South	East	West	Distance
State of environment	N S E W	N S E W	N S E W	N S E W	
Whitewasher rule	0 2 2 2	1 0 2 2	1 1 0 2	1 1 1 0	03
Painter rule	3 2 2 2	4 3 2 2	4 4 3 2	4 4 4 3	

Figure 2: Rule representation.

The rule structure is illustrated in Figure 2. The

last two digits of the whitewasher's rule represents the maximum distance allowed between the agents. The whitewasher holds its position if the actual distance between the painter and itself exceeds that number. The structure of the rules for the whitewasher and painter are the same otherwise.

The sensor function of an agent returns a pattern of four digits that represent the state of the surrounding environment, the four digits correspond to the directions North, South, East and West. If the pattern returned by the sensor functions matches the pattern represented by the first four digits of the agent's rule, the agent moves North; if it matches the second four digits, the agent moves South, etc. For example, consider the movement of the whitewasher when the sensor finds that the square to the North is whitewashed, the square to the South is empty, the square to the West is painted and the square to the East is empty. The first part of the whitewasher's rule in Figure 2, 0 2 2 2 is matched first. This part states that if the square to the North is empty (the squares at the South, West and East can be anything), the whitewasher moves North. The square to the North is whitewashed, and therefore, this rule does not match the state of environment. Next part of whitewasher's rule is 1 0 2 2. It says that if the square to the North is painted or whitewashed, and the square to the South is empty (the squares to the West and to the East can be anything), the whitewasher moves South. This rule matches the state of the environment, as the sensor has found that the square to the North is whitewashed and the square to the South is empty. The last two parts of the rules are ignored in this case. If none of the four digit patterns matches the pattern returned by the sensor function, the rule interpreter switches to a "global" mode. In the "global" mode, the sensor looks at every square in a specified direction up to the boundary of the room. For example, the first part of the rule in "global" mode implies that if there is an empty square anywhere directly to the North, move North.

A set of optimal rules (rules with which the two agents can paint all squares starting from the North-West corner) for this domain are the following:

Whitewasher:
0 2 2 2 1 0 2 2 1 1 0 2 1 1 1 0 0 3
Painter:
3 2 2 2 4 3 2 2 4 4 3 2 4 4 4 3

Figure 3 shows the movement of the agents using rules in Figure 2. Painter follows the Whitewasher and the maximum distance between the agents is 3. In the figure shown, the whitewasher moves first. The second part of the rule matches the output of the sensor function, which finds an empty square at the south. This

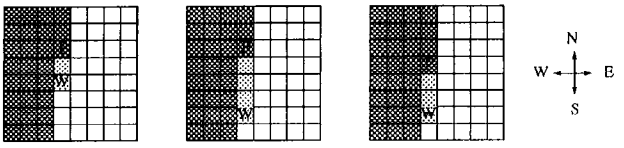

Figure 3: Example of room painting using rules in Figure 2.

results in the whitewasher moving south. The painter moves next. The sensor function finds a whitewashed square at the south. It matches the second part of the painter's rule. The painter moves south following the whitewasher.

Evaluation function. To get a fitness value for a pair, we run the simulator using a rule for the whitewasher and a rule for the painter. The simulator calculates the fitness value based on how much of the room is painted after a given number of ticks. Our objective is to maximize the number of squares painted. At the same time, we want to minimize the number of squares washed but not painted, because washed squares turn back empty if they are not painted within a specific time period, and it is a waste of effort to wash too many squares as the painter would not be able to catch up. We used the following evaluation formula in our experiments: $p-(w-p) = 2p-w$, where p is the number of squares painted and w is the number of squares washed. (w - p) is the number of squares washed but not painted. The maximum fitness value for a 8 x 8 room is 128 corresponding to all squares being painted.

Preliminary results and Future work

To evaluate the shared memory approach for this domain we ran two separate GAs: one for evolving whitewasher rules and another for evolving painter rules. We store pairs that performed best in shared memory. To get a fitness value for a whitewasher rule, we paired it with each of the painter rules stored in shared memory and ran the simulator on each of the pairs.

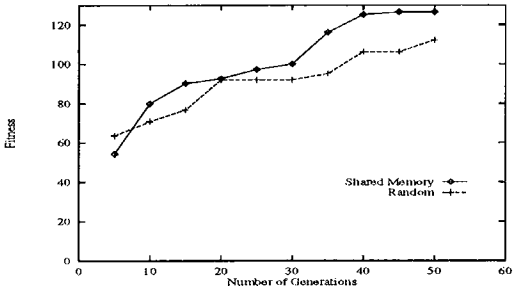

Figure 4: Shared Memory Vs Random pairing.

Figure 4 shows the online average (average of the

best individuals) of the populations over runs with different random seeds and using the following parameters: population size: 50, crossover rate: 0.9, mutation rate: 0.01, shared memory Size: 5, generations: 200. We ran two sets of 10 experiments each: one with random pairings and one with shared memory. We did not allow duplicate pairs in the shared memory. Results obtained from these preliminary experiments have been promising, with optimal rule pairs being consistently discovered by both approaches. The experiments showed that, while both the shared memory and the random technique generated close to optimal rule pairs, the former approach generates better rule pairs more consistently later in the runs.

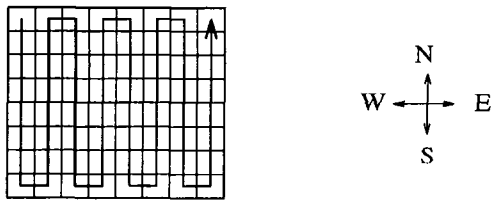

Figure 5: An example trace of the whitewasher and painter using a set of optimal rules generated by GA: 1102 1011 2221 2220 02 for whitewasher and 3223 4244 2434 2223 for painter.

All traces of solutions that demonstrate optimal or close to optimal performance are similar to the one in Figure 5. There are two reasons for that: (i) our rules have fixed order of preferences: North, South, West, East; (ii) we place both agents at position (0, 0) at the beginning of every simulation.

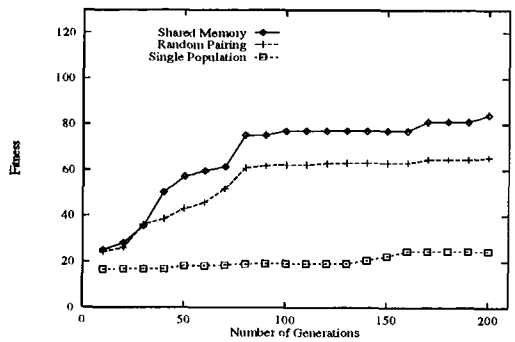

Figure 6: Performance curve using rules with the direction bits embedded.

The results described above were obtained by running GA's on agent rules without any direction genes. The order of considering the directions for moving were predefined (NSEW, in our experiments). This predefining of the direction placed a restriction on the movement of the agents. To be more specific, considering the rule representation used earlier, the agents always try to see if they can move north first. If the current world state does not favor this movement, the agents try to move towards the south, and so on. To remove this restriction placed on the agents, we decided to include direction genes as part of the agent rule. The direction genes represent the order in which the four directions are considered for possible movement. There are 4!=24 such orders and hence 3 ternary genes are used to represent the direction order used by an agent. We use a mapping from the 27 possible values represented by the 3 ternary numbers and the 24 possible orderings of the four directions. Consider the rules given in Figure 7. The direction genes in the whitewasher rule signifies that the agent always tries to move to the square which is to the west of the current position, failing which the agent tries to move to the east, followed by north and finally south. The painter rules signifies that this agent will try to move to the east initially, failing which the directions west, south and, north are considered in that order.

The experiments with the new rule representation were run using the following parameters: Population size: 50, Crossover rate: 0.9, Mutation rate: 0.05, Shared Memory Size: 5, Generations: 200. Given in Figure 6 are the performance curves obtained from these experiments. The curves represent the average fitness taken over 10 runs. Figure 7 also gives the trace of movements of the agents with the given rules. As shown in the figure, the last three genes in the rule determine the direction of movement of an agent. This arrangement allows the direction genes to be a part of the evolution process. This allows interesting co-adapted behavior to evolve which was not previously possible. We observed a variety of different movements resulting in a completely painted room. The movements depicted in Figure 7 was a very noticeable variation from that depicted in Figure 5 (that case was also observed in one of the runs in this second set of experiments).

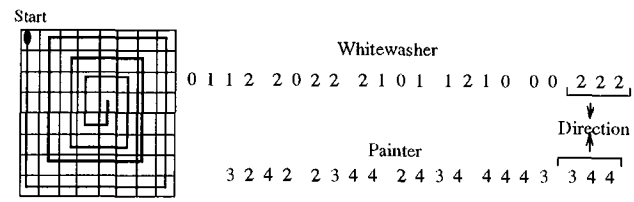

Figure 7: Trace obtained by using rules with direction bits.

It can be observed from Figure 6 that both the shared memory approach and the random approach

produce good pairs (though not as well as in the runs without directional genes; though the optimum result was found in a number of runs, a few bad runs significantly lowered the average). However, on an average, the shared memory approach gives better results. This performance enhancement using shared memory can be directly related to the fact that we use the best pairs from the previous generations to produce more individuals of high fitness in the generations to follow. On the contrary, in the random pairing approach, we simply rely on the performance of individuals in the current generation. Although pairs with good performance are produced, the number of pairs with high performance would be fewer than that obtained with the shared memory approach.

In addition to the experiments conducted using this new rule representation, we also conducted experiments to observe the behavior of the agents when their (whitewasher and painter) rules are combined into a single genome, i.e., a single population is used to evolve whitewasher-painter pairs. An example of such a genome is as follows:

0222 1020 2200 1120 01 001 | 4444 2322 4424 4443 432

In the above given genome, the first 21 genes correspond to the whitewasher rule, while the next 19 genes correspond to the painter rule. As mentioned earlier, the last three genes of each of the agent rules represent the direction. From Figure 6 it can be clearly seen that when the agents rules are combined into a single population, the performance is quite poor. This low performance may be caused because an agent does not have the option of pairing with a number of other individuals to obtain its fitness. Instead the agent must evolve along with same individual trying to improve their combined fitness.

Observations

From our experiments with directional genes, an interesting observation was the ability of the agents to co-adapt. It is to be noted from the example rule in Figure 7 that the direction genes for the whitewasher and the painter are distinct, signifying that they do not necessarily consider the directions in the same order. However, this does not affect their ability to adapt to each other and produce high fitness pairs. Another interesting observation was the trace (direction of movement) of the agents in painting the room. The use of direction genes in the agent rules resulted in many different traces, some of them rather surprising and interesting from our point of view. Figure 7 shows one such interesting trace of the agents.

We plan to expand our experiments to larger domains, and/or with more agents. We are currently conducting experiments by varying the shared memory size to study the effect of shared memory size on effective coevolution. We also plan to conduct experiments using the fitness proportionate scheme to select individuals from the other population to pair with (in place of the random pairing as described in this paper).

Acknowledgments:

This work has been supported, in part, by an NSF grant IRI-9410180.

References

Grefenstette, J. J., and Daley, R. 1996. Methods for competitive and cooperative co-evolution. In *AAAI-96 Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, 45–50.

Haynes, T., and Sen, S. 1996. Evolving behavioral strategies in predators and prey. In Weiss, G., and Sen, S., eds., *Lecture Notes in Artificial Intelligence*. chapter Adaptation and Learning in Multiagent Systems.

Haynes, T., and Sen, S. 1997. Crossover operators for evolving a team. to appear in the Proceedings of Genetic Programming 1997: Proceedings of the Second Annual Conference, 1997.

Haynes, T.; Sen, S.; Schoenefeld, D.; and Wainwright, R. 1995. Evolving a team. In *AAAI Fall Symposium on Genetic Programming*.

Ito, A., and Yano, H. 1995. The emergence of cooperation in a society of autonomous agents - the prisoner's dilemma game under the disclosure of contract histories. In *First International Conference on Multi-Agent Systems*, 201–208.

Potter, M. A.; Jong, K. A. D.; and Grefenstette, J. J. 1995. A coevolutionary approach to learning sequential decision rules. In *Sixth International Conference on Genetic Algorithms*, 366–372.

Rosin, C. D., and Belew, R. K. 1995. Methods for competitive co-evolution: Finding opponents worth beating. In *Sixth International Conference on Genetic Algorithms*, 373–380.