Evolving Behaviors for Cooperating Agents

Jeffrey K. Bassett and Kenneth A. De Jong

George Mason University
Computer Science Department
Fairfax, VA 22030
jbassett@cs.gmu.edu, kdejong@gmu.edu

Abstract. A good deal of progress has been made in the past few years in the design and implementation of control programs for autonomous agents. A natural extension of this work is to consider solving difficult tasks with teams of cooperating agents. Our interest in this area is motivated in part by our involvement in a Navy-sponsored micro air vehicle (MAV) project in which the goal is to solve difficult surveillance tasks using a large team of small inexpensive autonomous air vehicles rather than a few expensive piloted vehicles. Our approach to developing control programs for these MAVs is to use evolutionary computation techniques to evolve behavioral rule sets. In this paper we describe our architecture for achieving this, and we present some of our initial results.

1 Introduction

One of the most challenging aspects of building intelligent systems is the design and implementation of control programs for intelligent autonomous agents. Manually designing and implementing control programs that are sufficiently robust to handle dynamically changing environments and uncertainty has proved to be extremely difficult. As a consequence, there has been considerable interest in the use of machine learning techniques to help automate this process.

A good deal of progress has been made in this area in the past few years using a variety of representations (rules, neural nets, fuzzy logic, etc.) and a variety of learning techniques (symbolic, reinforcement, evolutionary, etc.). A natural extension of this work is to consider solving difficult tasks with teams of cooperating agents.

Our interest in this area is motivated in part by our involvement in a Navy-sponsored micro air vehicle (MAV) project in which the goal is to solve difficult surveillance tasks using a large team of small inexpensive autonomous air vehicles rather than a few expensive piloted vehicles. Our approach to developing control programs for these MAVs is to leverage off the successes in using evolutionary computation techniques to evolve behavioral rule sets for single-agent systems. In this paper we summarize related work, we describe our architecture, and we present some of our initial results. We conclude with a discussion of future work.

Z.W. Raś and S. Ohsuga (Eds.): ISMIS 2000, LNAI 1932, pp. 157-165, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2 Background

Our approach to developing teams of cooperating agents is to represent agent behaviors as sets of rules and evolve these rule sets using evolutionary computation techniques. There has been a good deal of work done in this area for single agents, but not cooperating teams of agents. At the same time, there has been work done on "collective robotics" using other techniques. In this section we summarize relevant work in these two areas.

2.1 Rule Learning using Evolutionary Algorithms

One of the earliest rule evolving approaches is Holland's classifier system [4]. In this system a population of rules is maintained. These rules both compete for space and priority, while also cooperating to produce an appropriate classification for the given input.

An alternative approach is to maintain a population of *rule-sets* which can vary in length. Examples of these are Smith's LS-1 system [9], and the GABIL system which uses a GA for concept learning [5]. Typically these types of systems build rules which have more of a stimulus-response quality.

The SAMUEL system [3] [8], arguably one of the more successful rule evolving systems, uses an interesting hybrid of these two approaches. Individuals are implemented as rule-sets, but SAMUEL also uses a rule bidding system and credit assignment mechanism similar to those found in a classifier system.

Wu, Schultz and Agah implemented a rule learning system for MAVs using a GA [11]. Their GA implementation was very much a canonical GA, with a binary representation, and proportional selection. Fitness was measured using a simulated environment, and each individual defined a variable length rule-set.

2.2 Collective Robotics

Collective robotics involves the use of robot teams which cooperate to perform a task or set of tasks [1]. Teams have several inherent advantages including the ability to distribute themselves, do problem decomposition, and perform parallel processing.

Robot soccer is one of the most popular domains for studying collective robotics. Tucker Balch implemented a soccer simulation to study task differentiation and specialization [2]. The robots were trained using Q-learning, and they would often specialize to playing either a defensive or offensive position.

Other problem domains include multi-robot box pushing [6], and foraging [7] tasks. These problems are often solved by implementing low level swarming behaviors such as *avoid* or *follow*. A learning algorithm is then used to teach the robots to select behaviors and coordinate with other robots.

A common problem in all these experiments was getting the robots to cooperate, particularly when learning algorithms were used. In each case the researchers found that evaluating individuals solely on their own performance wasn't enough. Only when the team was evaluated as a whole did cooperation occur.

3 Our EA Architecture

Our ultimate goal is to evolve heterogeneous team of specialized agents that collectively perform specific tasks. Our strategy for accomplishing this is to start simple and incrementally add complexity. Our first simplification is to assume that the teams consist of homogeneous agents, i.e., they are all executing the same task program. This allows us to focus on evolving a single program which, when simultaneously executed by a team of agents, produces collective cooperative behavior, and it allows us to take advantage of existing work on evolving single agent behaviors.

However, there are still a number of important design decisions that need to be made, such as how rule sets are represented internally in our EA, how rule sets are modified over time, etc. We discuss these design decisions in the following subsections.

3.1 Representation

In our architecture an individual in the population represents a complete set of rules, and its representation is a string in which all the rules are concatenated. The ordering of the rules is not important. From generation to generation, the length of individuals in the population will tend to vary in size. The system has parameters which define a minimum and maximum size for an individual, as well as an initial size.

Each rule is a fixed length binary string. Rules are composed of a condition clause and an action clause. The bits in the condition clause are mapped to the agent's sensors, while the bits in the action clause are mapped to the agent's actuators. This allows each agent to perceive its environment and take a corresponding action.

The rule interpretor used by each agent operates as follows. In any given situation, all the rules are compared to the current input from the sensors, and the rule that has the highest match score is executed. There are several possible ways of doing rule matching. For simplicity we have avoided using rule weights and bidding techniques such as in SAMUEL or classifier systems. Rule matching is described in more detail in the description of the agent environment.

3.2 Selection

Our population management scheme is different from a typical GA. We have implemented an ES-like model involving μ parents and λ offspring. Parent selection is deterministic: all individuals produce the same fixed number of offspring.

The selection bias in our architecture is implemented using survival selection. In an ES survivors are chosen in one of two ways: using a "+" strategy involving both the parent and child populations, or using a "," strategy involving only the child population. The former converges more rapidly but is more likely to find a local optimum, while the latter provides a broader but slower search. We have both options implemented, and experimentally choose the one best suited for the particular fitness landscape.

The ES community typically uses truncation selection for determining survivors. We have chosen to use a binary tournament instead because the selection pressure is weaker, allowing for more exploration early in the search.

3.3 Operators

Since the internal representation is binary, we use a standard bit-flip mutation operator. We also implemented both a 1-point and a 2-point crossover operator.

The 1-point crossover is the same operator as the one used by Wu et. al. [11]. Crossover can only occur on rule boundaries. Because individuals can vary in size, this crossover differs from the standard crossover operator used in most GAs. Instead of selecting crossover points at the same location on both parents, different crossover points are selected for each. This means that each child may contain either more or fewer rules than the parents which spawned them. In fact, this is the only mechanism by which rule sets can change in size.

The 1-point crossover operator does relatively little mixing of parental rules and does not produce any new rules. Based on earlier experience, we felt it would be useful to have a more disruptive crossover operator available as well. We chose to implement a 2-point crossover operator that was not restricted to crossing on rule boundaries. Crossover points are chosen by first picking random rule boundaries in both parents, just as with the 1-point crossover. Then a randomly chosen offset is applied to both crossover points to obtain the inter-rule cut point. This is essentially the same crossover used in the GABIL system [5].

3.4 Fitness

The fitness of a particular rule set is obtained via simulation. Agents within the simulation use the rule set to control their behaviors. The agents have a task to perform, and at the end of the simulation they are given a score which indicates how well the task was performed. Since our intention is to have these agents cooperate, they are all evaluated as a team, and all receive the same score. In the current implementation, all agents use the same rule set, and the resulting score from the simulator is used as the fitness of that rule set.

Without any sort of counteracting force, evolving rule sets tend to grow uncontrollably, very much the way Genetic Programs (GPs) do [10]. Parsimony pressure is used to discourage this growth by penalizing the fitness of larger individuals. We have implemented parsimony pressure with the same approach used by Wu, et. al. [11] as described by the formula $f'(i) = f(i) - \alpha l_i f(i)$. The interesting thing to note about this equation is that the penalty gets stronger as the raw fitness increases. This approach allows individuals to grow larger early in the process, perhaps improving the exploration phase of the search. The particular value used for α is experimentally determined.

4 Experimental Methods

4.1 Simulation

Currently all of our experiments involve the use of a simple micro air vehicle MAV simulator. The simulation environment is a 2-D arena surrounded by walls. Nine identical MAVs are placed into the simulator, and are allowed to move and turn on each timestep. The MAVs are like helicopters in that they can hover or move at a slow constant speed. As the MAVs move, they can potentially collide with each other or with the walls surrounding the arena. Any MAV that is involved in a collision is immediately destroyed and removed from the simulation.

Each MAV has 8 sonar sensors placed radially around the vehicle. These sensors have no range information. They return either a 0 or a 1, indicating whether or not there is an object in range in the direction the sensor is pointing. The sensor range can be adjusted as a parameter of the simulation.

The robots also have a surveillance range. They can "look" down and observe objects on the ground. Currently the MAVs pay no attention to what they are observing. Their only goal is to observe as much of the ground as possible at any given time.

The behavior of the MAVs is defined by a set of stimulus-response rules. Each rule is made up of 12 bits, and contains a condition and an action section. The first 8 bits are the condition section, with one bit for each sensor. At each timestep the current sensor readings are compared with all the conditions in the rule set. The rule with the closest match is the winner. If there is a tie, a winner is chosen randomly from among the best matches. There is also a minimum threshold for matches. At least half of the condition bits must match the current sensor configuration. If the winning rule exceeds this threshold, its action is executed.

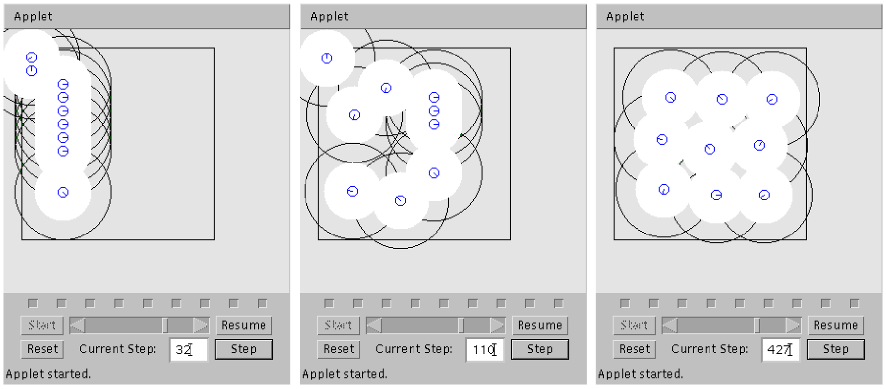

Fig. 1. MAV Simulator

The action section of the rule consists of two parts, a speed and a turn angle. The speed can have a value of either 0 or 1, where 0 indicates that the plane will not move in the current timestep, and 1 indicates that it will. The second part of the action is the turn angle. This indicates the number of degrees the plane will turn relative to its current heading.

Figure 1 provides a more concrete picture of an MAV simulation via a series of three snapshots from an example run involving a reasonably good set of evolved rules. The goal in this case is for a team of nine MAVS, starting from an initial configuration on the left edge of a surveillance area, to dynamically configure itself (without collisions!) in such a way as to obtain maximal surveillance coverage.

The simulator is stochastic in that the results of a simulation using the same rule set can change from run to run, resulting in a "noisy" fitness evaluation. Consequently, we typically run an individual through several trials. We then assign the average of all the trials as the fitness for the individual.

5 Initial Experimental Results

The goal of our initial experiments was to test our design decisions, tune our system, and evaluate its ability to evolve effective rule sets for teams of homogeneous agents. We describe these experiments in the following subsections.

The following parameters were used for our experiments unless stated otherwise. We set both μ and λ to 100. The number of trials was 5, while the crossover and mutation rates were 1.0 and 0.001 respectively. Individuals were limited to between 1 and 200 rules, with an initial starting size of 5 rules.

5.1 Population Management

Recall that we have both "+" and "," population management strategies available for use. The goal of our first set of experiments was to determine how sensitive the results are to this design choice. In general $(\mu + \lambda)$ slightly outperformed (μ, λ), although not by much. As a consequence we adopted the $(\mu + \lambda)$ strategy for the remaining experiments.

5.2 Parsimony Pressure

Another important design choice is the amount of parsimony pressure used. In general, too little parsimony pressure allows the length of individuals to grow indefinitely, and too much parsimony pressure produces compact individuals with suboptimal fitness. What is needed is a pressure point in between these two extremes. The goal of our second set of experiments was to get a rough sense of how parsimony pressure affected our system. We used three different values for parsimony pressure initially: 0, 1/2400 and 1/300. Figures 2 and 3 show that the parsimony pressure does work as expected. Higher parsimony pressures tend to produce smaller individuals, but at the expense of fitness.

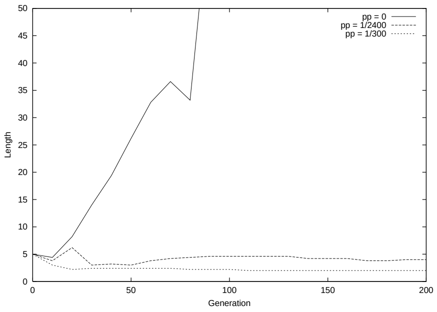

Fig. 2. Length of the best individual averaged over 5 runs for three parsimony pressures.

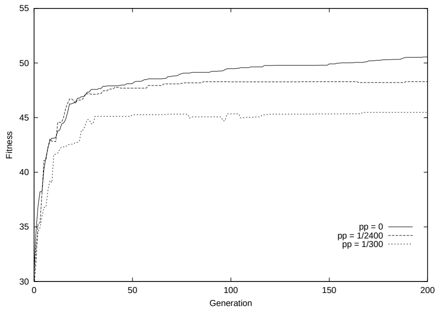

Fig. 3. Best-so-far curves of raw fitness averaged over 5 runs for three parsimony pressures.

5.3 Crossover

Recall that we have both a one and two point crossover operator implemented. The experiments so far have used the default 1-point crossover operator. Our third set of experiments involved testing the sensitivity of the results to these operators. Since we felt there could be some interaction with parsimony pressure, we tested sensitivity at a variety of pressure points: 0, 1/24000, 1/2400, 1/1200, 1/600 and 1/300. Five runs are performed for 200 generations at each parsimony pressure value.

In figure 4 we plot the raw fitness after 200 generations versus the parsimony pressure used. Again we see that higher parsimony pressures yield individuals with lower raw fitness values. However, we also see that 2-point crossover outperforms the 1-point crossover consistently at all levels of parsimony pressure. Consequently, we made 2-point crossover the default.

5.4 Generalization

Although our system is at this point evolving interesting and effective rule sets, figure 4 is somewhat disconcerting in that fitness declines steadily with increasing parsimony pressure. Ideally, one would hope to see shorter rule sets emerging with more general rules that achieve comparable performance. One possible explanation for why we don't see this is that the rule language itself is not well suited for generalization.

To test this we added classifier-like wildcards to our system by allowing the genes in the condition section of the rules to take on three values: 0, 1 and '*'. We also modified the random initialization and mutation operators so that we could adjust the number of wildcards in our individuals. We added a parameter called "wildcard ratio" which allows us to adjust the bias for the number of wildcards which end up in our rules. It can take a value between 0 and 1. A

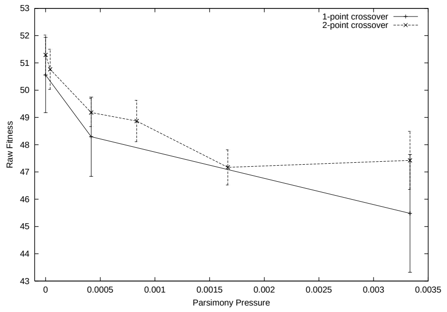

Fig. 4. Raw fitness vs. parsimony pressure is plotted for both the 1-point and 2-point crossover operators.

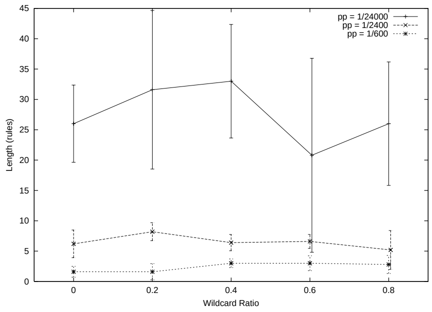

Fig. 5. Length vs. wildcard ratio bias is plotted using 3 different parsimony pressures: 1/24000, 1/2400, 1/600.

value of 0.4, for example, would mean that on average 40% of the genes in the condition sections of the rules will be a '*'.

We ran experiments using several values for the wildcard ratio and parsimony pressure. In figure 5 we plot length vs. wildcard ratio for the three different parsimony pressures. Each point on the graph represents the best individual at the end of 200 generations, and is an average of five runs.

As one can see, adding wildcards to the system had little effect on our ability to evolving smaller rule sets. A wild card ratio of zero is equivalent to having no wild cards. As we increase the wild card ratio the evolved rule lengths are basically unchanged regardless of the parsimony pressure. We see two possible explanations for this. First, our rule matching approach differs from the one used in classifier systems. We allow partial matches, and that acts as an alternative, and perhaps competing method of rule generalization. Another more likely explanation is that the problem domain we've chosen is just too simple. We believe that wildcards would be more useful if given a more difficult problem.

6 Conclusions and Future Work

We have completed our initial design and evaluation of an EA designed to evolve behavioral rules for teams of cooperating agents. Building on the work done for single agent systems, we were able to relatively quickly make and test design choices that resulted in the ability to evolve effective rule sets for teams of homogeneous agents. We are now continuing to develop the system further in several ways. First, we believe that the ability to evolve shorter and more general rule sets is important. Our initial experiments with wild cards were not successful. We are working on understanding this better.

Our ultimate goal is to work toward evolving heterogeneous cooperating agents. This initial work involving homogeneous provides a foundation for doing so, but needs further development. Extending our system to include notions of cooperative co-evolution seem quite appropriate here. We will be reporting on this in the near future.

Acknowledgments

This research was funded by a grant from the Naval Research Laboratory.

References

1. R.C. Arkin and T. Balch. Cooperative multiagent robotic systems. In David Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobil Robots*, Cambridge, MA, 1998. MIT/AAAI Press.
2. T. Balch. Learning roles: Behavioral diversity in robot teams. In *Collected Papers from the 1997 AAAI Workshop on Multiagent Learning*, pages 7–12, Cambridge, MA, July 1997. AAAI Press.
3. J. Grefenstette. Learning rules from simulation models. In *Proceedings of the 1989 International Association of Knowledge Engineers Conference*, pages 117–122, Washington, DC, 1989. IAKE.
4. J. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In J. Carbonell R. Michalski and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 593–623, Los Altos, 1996. Morgan Kaufman.
5. K. A. De Jong and W. M. Spears. Learning concept classification rules using genetic algorithms. In *IJCAI 91, Proceedings of the 12th International Conference on Artificial Intelligence*, pages 651–656, Sydney, Australia, 1991. Morgan Kaufmann Publishers, Inc.
6. C. Kube and H. Zhang. Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–219, 1993.
7. M. Mataric. Learning to behave socially. In D. Cliff, P. Husbands, J.A. Meyers, and S. Wilson, editors, *From Animals to Animats 3 (Third International Conference on Simulation of Adaptive Behavior)*, pages 453–462. MIT Press, 1994.
8. A. Schultz and J. Grefenstette. Using a genetic algorithm to learn behaviors for autonomous vehicles. In *Proceedings of the American Institute of Aeronautics and Astronautics Guidance, Navigation and Control Conference*, pages 739–749, Hilton Head, SC, 1992. AIAA.
9. S. Smith. Flexible learning of problem solving heuristics through adaptive search. In William Kaufman, editor, *Proceeding of the Eighth International Joint Conference on Artificial Intelligence*, pages 422–425, Karlsruche, Germany, 1983.
10. T. Soule and J.A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309, 1999.
11. A. Wu, A.C. Schultz, and A. Agah. Evolving control for distributed micro air vehicles. In *IEEE Computational Intelligence in Robotics and Automation Engineers Conference*, 1999.