Evolving a Team

Thomas Haynes, Sandip Sen, Dale Schoenefeld & Roger Wainwright
Department of Mathematical & Computer Sciences,
The University of Tulsa
e-mail: [haynes,sandip,dschoen,rogerw]@euler.mcs.utulsa.edu

Introduction

The goal of this research is to generate programs for the coordination of cooperative autonomous agents in pursuit of a common goal. In effect, we want to evolve behavioral strategies that guide the actions of agents in a given domain. The identification, design, and implementation of strategies for coordination is a central research issue in the field of Distributed Artificial Intelligence (DAI) [3]. Current research techniques in developing coordination strategies are mostly off–line mechanisms that use extensive domain knowledge to design from scratch the most appropriate cooperation strategy. It is nearly impossible to identify or even prove the existence of the best coordination strategy. In most cases a coordination strategy is chosen if it is reasonably good.

In [8], we presented a new approach for developing coordination strategies for multi–agent problem solving situations, which is different from most of the existing techniques for constructing coordination strategies in two ways:

- Strategies for coordination are incrementally constructed by repeatedly solving problems in the domain, i.e., on–line.
- We rely on an automated method of strategy formulation and modification, that depends very little on domain details and human expertise, and more on problem solving performance on randomly generated problems in the domain.

The approach proposed in [8] for developing coordination strategies for multi–agent problems is completely domain independent, and uses the strongly typed genetic programming (STGP) paradigm [13], which is an extension of genetic programming (GP) [11]. To use the STGP approach for evolving coordination strategies, the strategies are encoded as symbolic expressions (S–expressions) and an evaluation criterion is chosen for evaluating arbitrary S–expressions. The mapping of various strategies to S–expressions and vice versa can be accomplished by a set of functions and terminals representing the primitive actions in the domain of the application. Evaluations of the strategies represented by the structures can be accomplished by allowing the agents to execute the particular strategies in the application domain. We can then measure their efficiency and effectiveness by some criteria relevant to the domain. Populations of such structures are evolved to produce increasingly efficient coordination strategies.

We have used the predator–prey pursuit game [2] to test our hypothesis that useful coordination strategies can be evolved using the STGP paradigm for non–trivial problems. This domain involves multiple predator agents trying to capture a prey agent in a grid world by surrounding it. The predator–prey problem has been widely used to test new coordination schemes [5, 10, 12, 15, 16]. The problem is easy to describe, but extremely difficult to solve; the performances of even the best manually generated coordination strategies are less than satisfactory. We showed that STGP evolved coordination strategies perform competitively with the best available manually generated strategies.

The developed strategies had implicit communication in that the same program was used to control the four predator agents. In this work we examine the rise of cooperation strategies without implicit communication. This is achieved by having each predator agent being controlled by its own program. Such a system solves a cooperative co–evolution problem as opposed to a competitive co–evolution problem as described in [1, 7, 14].

We believe that cooperative co–evolution provides opportunities to produce solutions to problems that can not be solved with implicit communication.

Experimental Setup

In our experiments, the initial configuration consisted of the prey in the center of a 30 by 30 grid, and the predators are placed in random non–overlapping positions. All agents choose their action simultaneously. The environment is updated accordingly and the agents choose their next action based on the updated state. Conflict resolution is necessary since we do not allow two agents to co–occupy a position. If two agents try to move into the same location simultaneously, they are "bumped back" to their prior positions. One predator, however, can push another predator (but not the prey) if the latter decided not to move. The prey does not move 10% of the time: this effectively makes the predators travel faster than the prey. The grid is toroidal in nature, and diagonal moves are not allowed. A capture is defined as all four predator agents occupying the cells directly adjacent, and orthogonal, to the prey, i.e. when the predators block all the legal moves of the prey. A predator can see the prey, and the prey can see all the predators. Furthermore, two predators cannot communicate to resolve conflicts or negotiate a capture strategy.

Goals

We believe that laying down the framework for a cooperative co–evolution system will allow both for a team to learn together and for speeding up the learning process. The techniques that we describe can be applied to other GP based team problems. Specifically we will deal with the credit assignment problem of how to fairly split the fitness score to all participants in a team.

We believe that four different strategies for controlling the movements of an agent can be combined to form a cooperation strategy which will result in attaining a global goal. One contribution of this research will be to remove a level of communication in a DAI domain.

Establishing an Environment for Teamwork

In our earlier work, each program was represented as a chromosome in a population of individuals. The members of a team can randomly be selected from the population of chromosomes, with each member awarded a certain percentage of the total fitness. [1] Each member would get the points that it definitely contributed to the team's fitness score. How do we divide up the team's score among the participating members (chromosomes)? Is it fair to evenly divide the score? Assuming k members to a team, if the actions of one individual accounted for a large share of the team's score, why should it only get $\frac{1}{k}$th of the score? This problem is the same as the *credit assignment* problem in [6]. A modification of this strategy is to deterministically split the population into k sized teams. Thus the first k individuals would always form the first team. The problem with this is that it imposes an artificial ordering on the population. The same team in generation G_i might not be formed in generation G_{i+1} due to a re–ordering caused by the reproductive cycle.

The method we employ to ensure consistency of membership of a team is to evolve a team rather than an individual. Thus each chromosome represents k programs. Subject to the effects of crossover and mutation, we are ensured that the same members will form a team. This effectively removes the credit assignment problem. Each team member always participates in the same team. Thus all of the points it is awarded, for both its individual contribution and the teams contribution, are correctly apportioned to the entire team.

This approach is similar to "the Pitt approach" used for evolving Genetic–Based Machine Learning systems [4]. For GA based production systems, there are two camps as how to maintain a ruleset: the Pitt approach is to maintain the entire ruleset as an individual string with the entire population being a collection of rulesets, and "the Michigan approach" is to maintain the entire population as the ruleset. In the Michigan approach there is the credit assignment problem of how to correctly award individual rules for

[1] We could also ensure that each member of the population participates in t teams.

their contributions to the global solution. The Pitt approach bypasses the credit assignment problem, in that rules are only evaluated in the context of a ruleset. A similar mechanism as proposed in this paper has been used to successfully co–evolve a set of prototypes for supervised concept classification problems [9].

Our method of maintaining consistency in a team does introduce a problem in that what do we do for crossover? Do we allow crossover, as shown in Figure 1, to take place in the usual sense? (i.e. only one of the programs participates in the crossover.) Or, as shown in Figure 2, do we allow all of the programs to participate in crossover? The first crossover mechanism allows only relatively small changes of parent structures to produce offspring, and thus slows down learning.

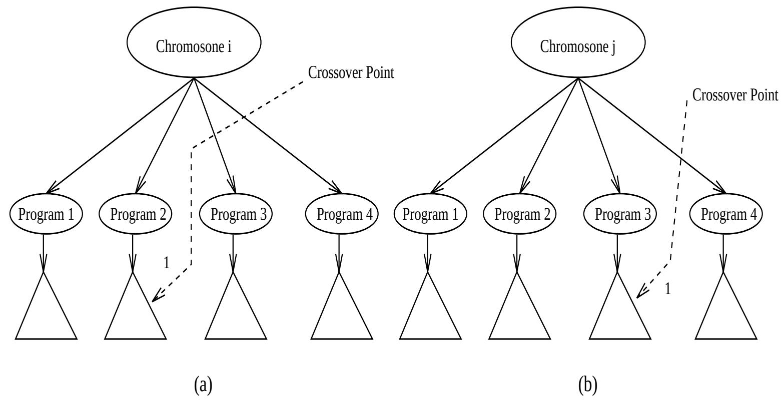

Figure 1: Example crossover for 1 crossover point in a chromosome.

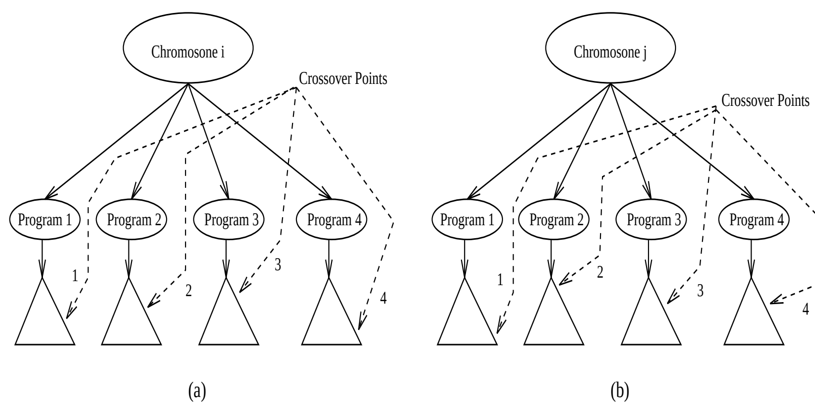

Figure 2: Example crossover for all programs in a tree. A crossover point is selected in the subtree of each program. Thus there are four crossovers taking place; between each program P_i for the two chromosomes.

The second crossover mechanism will speed up the emergence of good cooperation strategies by allowing each program in a parent structure to participate in the crossover process. A research issue in this crossover method is determining whether we should constrain crossover between corresponding programs in the two parents. If the first program in the first parent always crosses over with the first program in the second parent, then can the first program become a specialist? There can be a need for specialists, i.e. the dessert maker in a team of cooks, but in applying this constraint we restrict do we restrict ourselves to a part of the solution space in which the global optimum can not be found?

Some possible solutions to this concern are:

1. For chromosomes A and B, randomly determine which program A_i will be used in crossover with program B_j. Also each program in a chromosome participates exactly once in the crossover process.
2. A new mutation operator could be defined which swaps subtrees between programs in a chromosome.

A third crossover mechanism is to adapt the uniform crossover function from GA research. Basically we would develop a uniform crossover mask for the programs inside a chromosome. A "1" would indicate that the programs are swapped, while a "0" would indicate that the programs would undergo crossover. We are able to use the uniform crossover function because the number of programs in a team is fixed. Since the programs are not atomic in the sense that alleles in GAs are, we could randomly determine the interactions between the programs. An example of this is if we decided that the order of interaction between two parent chromosomes i and j is $i(3241)$ and $j(4123)$, and the bit mask is $\{1001\}$, then this would produce the children $s(3(2X1)(4X2)1$ and $t(4(2X1)(4X2)3$. This is represented visually in Figure 3. The programs have been re–ordered such that $i3$ is paired with $j4$, etc.

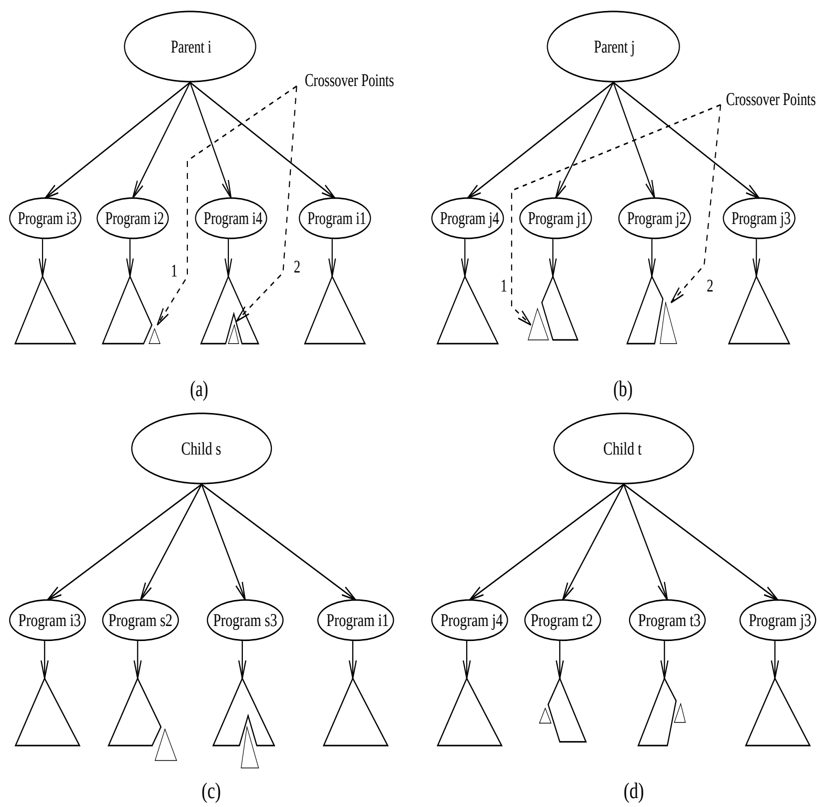

Figure 3: Example uniform crossover for the mask (1001). (a) has Parent i with an ordering of (3241). (b) has Parent j with an ordering of (4123). (c) has Child s, with two children created via crossover. (d) has Child t, with two children created via crossover.

A fourth crossover function is to allow k crossover points inside a chromosome. A restriction is that crossover point i can not be an ancestor node of any crossover point $j, j \neq i$. A difference between this method and the previous methods is that two crossovers can happen to the same program, as can be seen in Figure 4. Each crossover point i is not tied to any one program.

Status of Research

We are currently conducting experiments with the second method of representing a team, i.e. where the four programs are stored in one chromosome. In our previous research, we found that several days were needed per learning session. This has been compounded by the need to evolve four programs per chromosome.

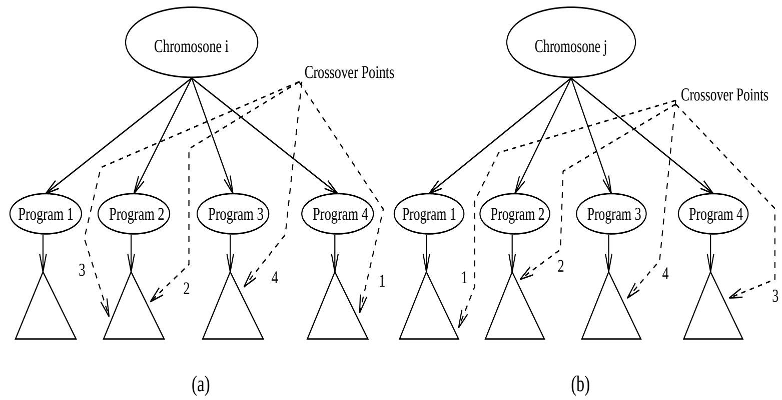

Figure 4: Example crossover k crossover points in a chromosome.

References

[1] Peter J. Angeline and Jordan B. Pollack. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 264–278. Morgan Kaufmann Publishers, Inc., 1993.

[2] M. Benda, V. Jagannathan, and R. Dodhiawalla. On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center, Boeing Computer Services, Bellevue, WA, August 1985.

[3] Alan H. Bond and Les Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[4] Kenneth A. DeJong. Genetic-algorithm-based learning. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning, Volume III*. Morgan Kaufmann, Los Alamos, CA, 1990.

[5] Les Gasser, Nicolas Rouquette, Randall W. Hill, and John Lieb. Representing and using organizational knowledge in DAI systems. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*, pages 55–78. Pitman, 1989.

[6] John Grefenstette. Credit assignment in rule discovery systems. *Machine Learning*, 3(2/3):225–246, 1988.

[7] Thomas Haynes and Sandip Sen. Evolving behavioral strategies in predators and prey. In *IJCAI-95 Workshop on Adaptation and Learning in Multiagent Systems*, 1995.

[8] Thomas Haynes, Roger Wainwright, Sandip Sen, and Dale Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995. (accepted for publication).

[9] Leslie Knight and Sandip Sen. Please: A prototype learning system using genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995. (accepted for publication).

[10] Richard E. Korf. A simple solution to pursuit games. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 183–194, February 1992.

[11] John R. Koza. *Genetic Programming, On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[12] Ran Levy and Jeffrey S. Rosenschein. A game theoretic approach to the pursuit problem. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 195–213, February 1992.

[13] David J. Montana. Strongly typed genetic programming. Technical Report 7866, Bolt Beranek and Newman, Inc., March 25, 1994.

[14] Craig W. Reynolds. Competition, coevolution and the game of tag. In *Artificial Life IV*. MIT Press, 1994.

[15] Larry M. Stephens and Matthias B. Merx. Agent organization as an effector of dai system performance. In *Working Papers of the 9th International Workshop on Distributed Artificial Intelligence*, September 1989.

[16] Larry M. Stephens and Matthias B. Merx. The effect of agent control strategy on the performance of a DAI pursuit problem. In *Proceedings of the 1990 Distributed AI Workshop*, October 1990.