Crossover Operators for Evolving A Team

Thomas Haynes and Sandip Sen
Department of Mathematical & Computer Sciences
600 South College Ave.
The University of Tulsa
Tulsa, OK 74104-3189
e-mail: [haynes,sandip]@euler.mcs.utulsa.edu

Abstract

Cooperative co–evolutionary systems can facilitate the development of teams of heterogeneous agents. We believe that k different behavioral strategies for controlling the actions of a group of k agents can combine to form a cooperation strategy which efficiently achieves global goals. We examine the on–line adaption of behavioral strategies utilizing genetic programming. Specifically, we deal with the credit assignment problem of how to fairly split the fitness of a team to all of its participants. We present several crossover mechanisms in a genetic programming system to facilitate the evolution of more than one member in the team during each crossover operation. Our goal is to reduce the time needed to evolve a good team.

1 Introduction

We have utilized genetic programming (GP) [Koza, 1992] to evolve behavioral strategies which enabled a team of loosely–coupled agents to cooperatively achieve a common goal [Haynes and Sen, 1996, Haynes *et al.*, 1995]. Since they each shared the same behavioral strategy, the agents were homogeneous, i.e., each chromosome in the population implicitly represented the program of k agents. A simple algorithm to model the actions of others is to believe that they behave as you would in the same situation. With homogeneous agents, the agents can employ this algorithm since their models of other agents matches the actions of others. A key issue in distributed artificial intelligence (DAI) research is how can heterogeneous agents interact to form a team. We extend our research from the evolution of homogeneous agents to the evolution of heterogeneous agents in a team. In this paper, each chromosome in the population explicitly represents k programs, each corresponding to an agent. While we expect a degradation of performance from agents whose models of other agents are wrong, i.e., the agents are heterogeneous while the models are homogeneous, we also expect to see a rise in effectiveness of the cooperation between team members. As our system does not allow an agent to change its models of the other members, we believe that team members will evolve into near homogeneity to facilitate efficient group behavior.

We test our heterogeneous team in the predator–prey game, a testbed domain from the DAI field. Four predator agents try to capture a prey agent by surrounding it orthogonally. The agents exhibit symmetry in that the predators are interchangeable. Animal hunters can adopt roles in the hunt: one scouts out the quarry, one flushes the quarry, and another kills the quarry. The roles are dynamically allocated to fit the state of the current hunt; each hunter must be able to perform each role.

2 Coordination Strategies

We presented a new approach for developing coordination strategies for multiagent problem solving situations [Haynes *et al.*, 1995], which is different from most of the existing techniques for constructing coordination strategies in two ways: 1) Strategies for coordination are incrementally constructed by repeatedly solving problems in the domain, i.e., on–line.; and 2) 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.

Our approach uses strongly typed genetic programming (STGP) [Montana, 1995] to evolve the 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 [Benda *et al.*, 1986] to test if useful coordination strategies can be evolved using the STGP paradigm for non–trivial problems. This domain involves multiple predator agents trying to capture a mobile prey agent in a grid world by surrounding it. The predator–prey problem has been widely used to test new coordination schemes [Korf, 1992, Stephens and Merx, 1990]. 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. STGP evolved coordination strategies performed competitively with the best available manually generated strategies.

In this work we examine the rise of cooperation strategies without implicit communication. In our previous research, the developed strategies had implicit communication in that the same program was used to control the predator agents. This removal of implicit communication 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 [Angeline and Pollack, 1993, Haynes and Sen, 1996, Reynolds, 1994]. We believe that cooperative co–evolution provides opportunities to produce solutions to problems that cannot be solved with implicit communication.

3 Pursuit Domain

In our experiments, the initial configuration consists 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. For the training cases, each team is allowed 100 moves per case. The environment is updated after all of the agents select their moves, and then the agents again 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's movements are controlled by a strategy that moves it away from the nearest predator, with all ties being non–deterministically broken. 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. However, two predators cannot communicate to resolve conflicts or negotiate a capture strategy. The latter eliminates explicit communication between agents.

To evolve coordination strategies for the predators using STGP we need to rate the effectiveness of those strategies represented as programs or S–expressions. We chose to evaluate such strategies by putting them to task on k randomly generated pursuit scenarios. For each scenario, a program is run for 100 time steps. The percentage of capture is used as a measure of fitness when we are comparing several strategies over the same scenario. Since the initial population of strategies are randomly generated, it is very unlikely that any of these strategies will produce a capture. Thus we need additional terms in the fitness function to differentially evaluate these non–capture strategies. The key aspect of GPs (including STGP) or GAs is that even though a particular structure is not effective, it may contain useful substructures which when combined with other useful substructures, will produce a highly effective structure. The evaluation (fitness) function should be designed such that useful sub–structures are assigned due credit.

With the above analysis in mind, we designed our evaluation function of the programs controlling the predators to contain the following terms:

- After each move is made according to the strategy, the fitness of the program representing the strategy is incremented by (Grid width) / (Distance of predator from prey), for each predator. Thus higher fitness values result from strategies that bring the predators closer to the prey, and keep them near the prey. This term favors programs which produce a capture in the least number of moves.

- When a simulation ends, for each predator occupying a location adjacent to the prey, a number equal to (number of moves allowed $*$ grid width) is added to the fitness of the program. This term favors situations where one or more predators surround the prey.

- If a simulation ends in a capture position, an additional reward of (4 $*$ number of moves allowed $*$ grid width) is added to the fitness of the program. This term strongly biases the evolutionary search toward programs that enable predators to maintain their positions when they succeed in capturing a prey.

In our experiments, the distance between agents is measured by the *Manhattan distance* (sum of x and y offsets) between their locations. We have limited the simulation to 100 time steps. As this is increased, the capture rate will increase. In order to generate general solutions, (i.e., solutions that are not dependent on initial predator–prey configuration), the same k training cases were run for each member of the population per generation. The fitness measure becomes an average of the training cases. These training cases can be either the same throughout all generations or randomly generated for each generation. In our experiments, we used random training cases per generation.

4 An Environment for Teamwork

In our earlier work, each program was represented as a chromosome in a population of individuals. One method to compose a team from different chromosomes is to randomly selected members from the population of chromosomes, with each member awarded a certain percentage of the total fitness. (We could also ensure that each member of the population participates in t teams.) 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 [Grefenstette, 1988]. Another partitioning 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 consists of 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 [DeJong, 1990]. 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 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.

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. We present several different mechanisms to allow multiple programs to participate during the crossover process:

TeamTree For comparison purposes we present the method in which all agents share the same program.

TeamBranch This method is simply to pick one crossover point in the chromosome (see Figure 1). This would be the traditional GP crossover mechanism.

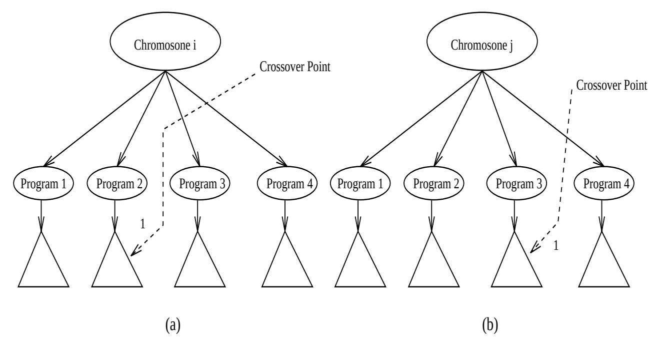


Figure 1: Example crossover for 1 crossover point in a chromosome.

TeamAll This 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 (see Figure 2). 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, e.g., the dessert maker in a team of cooks, but in applying this constraint do we restrict ourselves to a part of the solution space in which the global optimum can not be found?

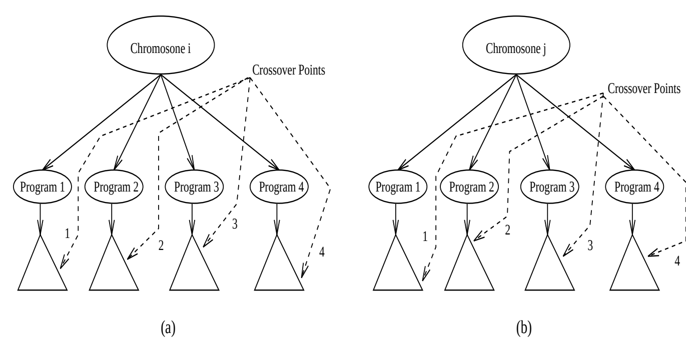


Figure 2: Example crossover for all programs in a tree. A crossover point is selected in the subtree of each program. Thus there are four crossovers taking place; between each program P_i for the two chromosomes.

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.; and 2) A new mutation operator could be defined which swaps subtrees between programs in a chromosome. This is different than recombination in that there is only one "parent" and one resultant "child".

TeamAll-Random This crossover mechanism is to randomly select the program A_i will be used in crossover with program B_j in the TeamAll method. (The rest of the crossover mechanisms adopt the random selection of parents within the context of the desired methodology.)

TeamUniform This crossover mechanism is to adapt the uniform crossover function from GA research (see Figure 3). Basically we would develop a uniform crossover mask for the programs inside a chromosome. A "1" would indicate that the programs are copied into the respective child, 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.

TeamKCross This crossover mechanism is to allow k crossover points inside a chromosome (see Figure 4). 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.

5 Related Work

Angeline has investigated adaptive crossover operators for single–branched chromosomes [Angeline, 1996]. The self–adaptive multi–crossover (SAMC) adaptively determines both the number and position of crossover points in each chromosome. The operators we report differ in that the number of crossover points is a function of the number of branches in the chromosome and their positions in the chromosome are random.

6 Results

In a series of experiments, we have evaluated the different crossover mechanisms for evolving teams comprised of heterogeneous agents. The basic setup for each experiment was a population size of 600, a maximum of 1000 generations, and a maximum fitness of 48,000. In each generation, each chromosome is evaluated against the same three random initial placements. We ran each approach with the same six different initial seeds for the random number generator. The averaged results for the Best and Average Fitnesses per generation for the different crossover methods are shown in Figures 5– 10. We can rank the methods as:

1) TeamUniform
2) TeamTree, TeamBranch
3) TeamAll, TeamAll-Random

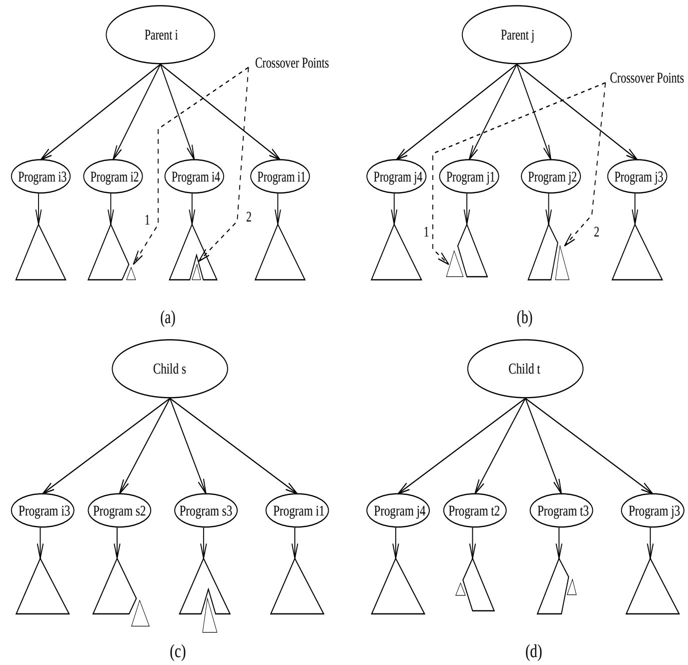


Figure 3: Example uniform crossover for the mask (1001). (a) has Parent i with an ordering of (3241). (b) has Parent j with an ordering of (4123). (c) has Child s, with two children created via crossover. (d) has Child t, with two children created via crossover.

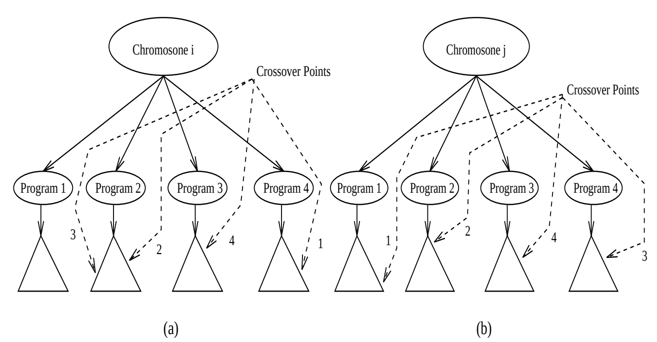
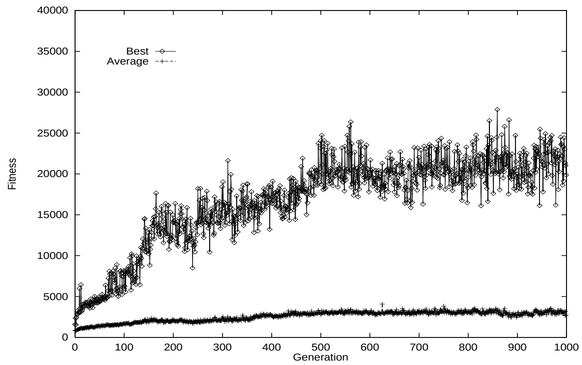


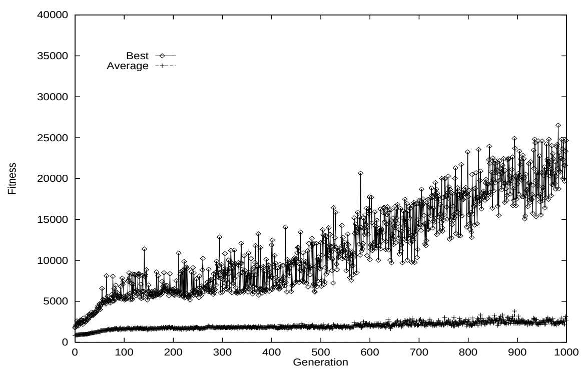
Figure 4: Example k crossover points in a chromosome.

4) TeamKCross



(a)

Figure 5: Average and Best Fitness for TeamTree.

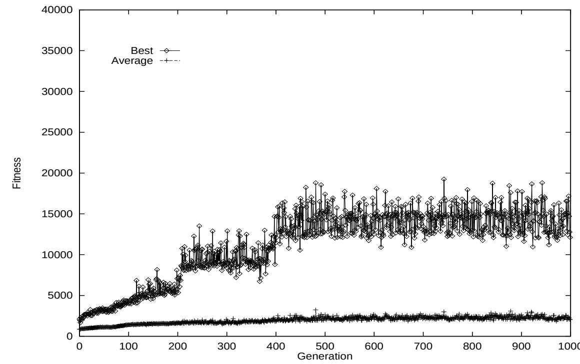


(b)

Figure 6: Average and Best Fitness for TeamBranch.

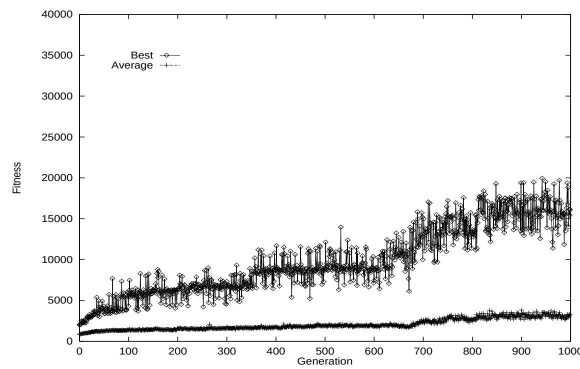
From Figures 5 and 6 we see that TeamTree learns faster than TeamBranch. This is due to the implicit communication in the form of homogeneity that the agents in TeamTree employ. As mentioned, this facilitates simple modeling of others. What we find surprising is that only one of the four other crossover methods, TeamUniform, learns to cooperate better than TeamBranch.

In examining the movements of the TeamUniform agents, we realized one of the benefits of heterogeneous predator agents: they are able to move in different directions when in the same quadrant with respect to the prey's orthogonal axis. One of the observed behaviors in both the evolved homogeneous and hand-crafted behavioral strategies is that if two predators were in the same quadrant, then they would select the same action. This behavior would lead to deadlock situations, for example if predators **1** and **2** are lined up on the horizontal axis with respect to the prey **P**, then one of the two predators cannot get to a capture position. With the heterogeneous behavioral strategies, deadlock situations have the potential to be avoided.



(c)

Figure 7: Average and Best Fitness for TeamAll.



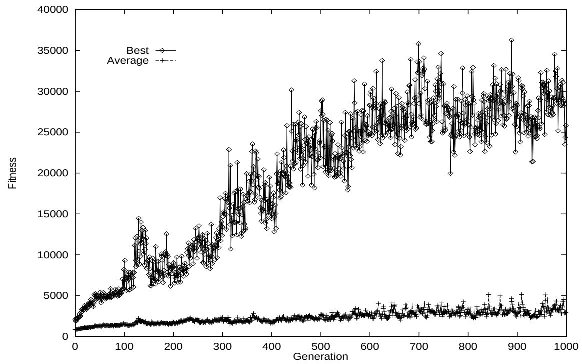
(d)

Figure 8: Average and Best Fitness for TeamAll-Random.

From our analysis of the best four teams per crossover system, we determined that the some of the TeamBranch and TeamAll behavioral strategies allow the prey to escape capture. This does not happen when the same strategy is used to control all agents, i.e. TeamTree (or STGP from our previous research). From the best TeamAll strategy, we find that two of the predator agents evolve very similar movement strategies, suggesting that the predator agents are learning the same behavioral strategy (i.e. becoming homogeneous), which in turn implies implicit communication is starting to take place. A similar occurrence of this duplication of strategies was observed in one of the four best TeamBranch chromosomes. Note that these graphical representations do not capture the dynamic interactions caused by a moving prey.

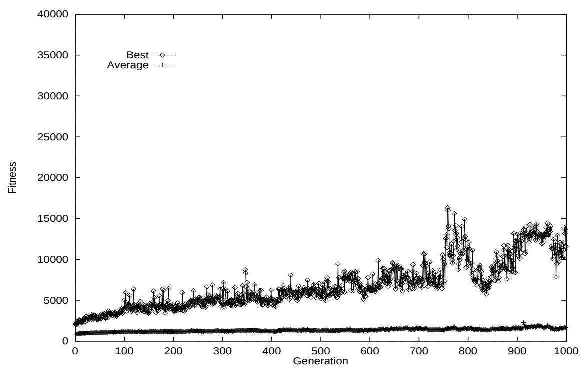
7 Conclusions

We have introduced four different crossover mechanisms for chromosomes containing more than one result producing branch. One of the crossover mechanisms, i.e. TeamUniform, was found to both speed up the evolutionary process and attain higher fitness than the traditional



(e)

Figure 9: Average and Best Fitness for TeamUniform.



(f)

Figure 10: Average and Best Fitness for TeamKCross.

GP crossover mechanism. We believe that the uniform crossover method will also benefit other domains with more than one executable branch in the chromosome. In particular, it could be of benefit to a genetic programming system employing ADFs.

We have also found that heterogeneous agents have been able to excel in a symmetrical domain. At first we thought that heterogeneous agents would suffer from the lack of simple models of others (A capability which can be employed in homogeneous agent systems.). But we found that if heterogeneous agents are presented with essentially the same input, i.e., a similar state induced by symmetry, they can perform different actions. This asymmetry of behavioral strategies allows the agents to avoid potential deadlock situations.

References

[Angeline and Pollack, 1993] 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.

[Angeline, 1996] Peter J. Angeline. Two self-adaptive crossover operators for genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 5, pages 89–110. MIT Press, Cambridge, MA, 1996.

[Benda *et al.*, 1986] 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, WA, July 1986.

[DeJong, 1990] 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.

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

[Haynes and Sen, 1996] Thomas Haynes and Sandip Sen. Evolving behavioral strategies in predators and prey. In Gerhard Weiß and Sandip Sen, editors, *Adaptation and Learning in Multi–Agent Systems*, Lecture Notes in Artificial Intelligence, pages 113–126. Springer Verlag, Berlin, 1996.

[Haynes *et al.*, 1995] Thomas Haynes, Roger Wainwright, Sandip Sen, and Dale Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 271–278, San Francisco, CA, 1995. Morgan Kaufmann Publishers, Inc.

[Korf, 1992] 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.

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

[Montana, 1995] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[Reynolds, 1994] Craig W. Reynolds. Evolution of obstacle avoidance behavior: Using noise to promote robust solutions. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pages 221–241. MIT Press, Cambridge, MA, 1994.

[Stephens and Merx, 1990] 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.