Co–adaptation in a Team

Thomas D. Haynes and Sandip Sen
600 South College Ave.
Department of Mathematical & Computer Sciences,
The University of Tulsa
Tulsa, OK 74104-3189 USA

Research partially supported by NSF Research Initiative Award IRI–9410180.

This is a preprint of an article that is going to appear in *International Journal of Computational Intelligence and Organizations (IJCIO)* sometime late 1996 or early 1997.

Abstract

We introduce a cooperative co–evolutionary system to 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 both examine the on–line adaption of behavioral strategies utilizing genetic programming and demonstrate the successful co-evolution of cooperative individuals. We present a new crossover mechanism for genetic programming systems in order 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 an effective team.

1 Introduction

Russell and Norvig (Russell and Norvig, 1995) define an agent as anything which can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. Insects, animals, robots, and humans all clearly fall into this definition. We can also consider software programs to be agents; the environment is the operating system and system calls are sensors and effectors. With the transition of the Internet to the Information Superhighway, we are entering an age where the information available at our fingertips exceeds our capacity to process it. The Internet is ever growing and new protocols are developing. (These protocols include proprietary databases, web page layouts, etc.) Just as we cannot hope to keep up with the explosive growth of the Internet, we cannot expect one software agent to handle all of the emerging data formats.

What we can envision is a society of interacting agents, exchanging information via a standard language, such as KQML (Barbuceanu and Fox, 1995; Finin et al., 1994). (Such dialog could be free or subject to market pressures.) Agents would be able to translate a data format, integrate knowledge from other agents, broker services, etc. Several surveys have been made into how human organizational theory can be leveraged into computational agent societies (Fox, 1981; Malone, 1987; Malone, 1990; Mullen and Wellman, 1995). The present work compliments this body of literature by providing a novel use of evolutionary algorithms for generating a team of co–adapted agents that perform effectively in their environment.

Coordination of the actions of agents is one of the key research topics in multiagent systems (MAS) (Lesser, 1995; Malone, 1987; Shaw, 1996). Distributed artificial intelligence (DAI) researchers study both organizational behavior (Fox, 1981; Malone, 1987; Robbins, 1993) and adaptive behavior in nature (Gotwald, Jr., 1995; Krebs and Davies, 1993) to gain insight as to how groups effectively interact during problem solving. We are interested in how groups form, maintain common purpose, and learn how to adapt their individualistic goals to a common group goal. In this research we consider the latter problem and investigate a domain in which simple greedy behavior must be adapted to cooperative group behavior. Such mutual adaptation and learning is of paramount importance to designers of agent groups and societies (Weiß and Sen, 1996; Sen, 1996).

Genetic programming (GP) (Koza, 1992) is an offshoot of genetic algorithms (GA) (Holland, 1975), and effectively performs an implicit parallel search through the problem space. A population composed of random programs, or chromosomes, are constructed out of a domain specific language. Each chromosome can be evaluated by a domain specific fitness function. Chromosomes are then selected for the next generation, with higher scoring chromosomes being more likely to be picked. Chromosomes undergo reproduction in a process similar to that seen in biological systems.

We have utilized genetic programming 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., 1995b). Since the agents shared the same behavioral strategy, they were homogeneous, i.e., each population member 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 (Haynes et al., 1996). With homogeneous agents, the agents can employ this algorithm since their models of other agents matches the actions of others. A key issue in DAI research is how can heterogeneous agents cooperate to form a successful team. In this paper, 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.

Our core problem is that of credit assignment, i.e., how to fairly reward an individual agent for its contributions to the group goal. When individual agents perform independent tasks in a shared environment, it is relatively simple to distribute credit. As the goals of the individual start to overlap those of the group, the distribution of credit becomes problematic: if it is the interaction of two individuals that achieves the goal, how do we determine the distribution of credit? Suppose the first agent did nothing to aid the second, its presence was sufficient for the task to be done. Should both agents get an equal share of the credit? What if by the first agent helping just a little, the task was done faster or cheaper? Should the first agent receive equal credit in both scenarios?

We address this problem by evolving individuals in the context of a team. The credit goes not to an individual, but to the team as a whole. Thus the learning takes place in the team, but the adaptation occurs in individuals. Consider a scenario where one member of a team of k individuals is significantly more effective than the others in realizing the team's goals. If the team performed well, the underlying genetic algorithm will most likely select it to undergo recombination into the next generation. The single agent is not rewarded directly. When the team undergoes recombination with another team, individual members are stochastically selected, without regard to contribution to the team's goals, for adaptation. Either the effective member aids the other team's learning or the ineffective members can engage in learning from the other team. On the average, over time the ineffective members will adapt to the level of the effective one.

While our proposed approach is a novel way to evolve a strategy for coordinating a group of agents, the process of generating the strategy is not distributed in nature. That is, agent teams are evolved as a whole and there is no individual learning being performed by each agent. Even though individual agents do not control the adaptation process, since the GP is concurrently adapting the team members, we believe "co-adaptation" is the proper term to describe the evolutionary process. Once endowed with the evolved behaviors, each agent is completely autonomous in its interactions with both the environment and other team members, as there is no centralized control.

We use the predator/prey or pursuit problem as a testbed for our experiments with evolving heterogeneous agent groups. Though this problem does not provide all of the possible challenges for heterogeneous groups, it is sufficiently complex and provides a good starting point. The predator–prey game is a well studied stylized domain which researchers in multiagent systems have utilized to evaluate organizations, control structures and messaging systems (Stephens and Merx, 1989; Stephens and Merx, 1990). The goal is for four predator agents to capture a prey agent by surrounding it orthogonally.

Although all the agents in this domain have the same capabilities, in a given pursuit different agents might be required to play different group roles. Consider the analogous situation of a team of hunters chasing a target. 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 capable of performing each role.

The key aspects of the domain are that agent groups must be effectively coordinated while retaining individual autonomy and that groups must be able to anticipate and adapt their problem solving technique to new and demanding situations. An off–line design of behavioral strategies for the agents is bound to be limited in its applications as all possible interactions cannot be anticipated at design time. Thus we are interested in on–line adaptive mechanisms for tailoring group behavior.

Agents, and in particular, groups of agents are becoming increasingly commonplace in the real world. Although we use an artificial problem domain, the methodology we investigate is clearly promising enough to address broader challenges in the future. Genetic algorithms were once much maligned for working with benchmark problem suites, but nowadays almost all applications are in engineering fields (Davis, 1991). Our work will show the potential for using GPs in developing effectively coordinated multiagent systems. We plan to further investigate real–life agents like coordinated meeting schedulers and cooperative controllers with the proposed methodology. But we have to develop the methodology using a well-understood problem domain which can be effectively simulated and in which we have enough data available to compare our methodology with that of other techniques that have been used in the domain. The predator-prey domain perfectly suits this bill.

The rest of this paper is laid out as follows: Section 2 presents our methodology for coordinating agent groups. Section 3 introduces the pursuit domain. Section 4 provides an overview of genetic programming, presents how we evaluate predator–prey games, and describes the experimental setup. Section 5 presents our crossover strategy for improving the learning of the team. Section 6 compares the utility of the crossover strategies as they evolve teams. Section 7 wraps up our research into team formation. Section 8 points out how this work can be extended.

2 Coordination of Agent Groups

The underlying goal of our research is to generate programs for the coordination of cooperative actions from a group of autonomous agents which are initially uncooperative. The agents must adapt from greedy local behavior to working towards 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 DAI (Bond and Gasser, 1988). 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. In most cases a coordination strategy is chosen if it is reasonably good.

In (Haynes et al., 1995b), we presented a new approach for developing coordination strategies for multiagent 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.

Our approach for developing coordination strategies for multi–agent problems is completely domain independent, and uses the strongly typed genetic programming (STGP) paradigm (Montana, 1995), which is an extension of GP. 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.

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

We have used the predator–prey pursuit game (Benda et al., 1986) 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 mobile prey agent in a grid world by surrounding it. The predator–prey problem has been widely used to test new coordination schemes (Gasser et al., 1989; Korf, 1992; Levy and Rosenschein, 1992; Stephens and Merx, 1989; 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. We showed that STGP evolved coordination strategies perform competitively with the best available manually generated strategies.

The original version of the predator–prey pursuit problem was introduced by Benda, *et al.* (Benda et al., 1986) and consisted of four blue (predator) agents trying to capture a red (prey) agent by surrounding it from four directions on a grid–world. Agent movements were limited to either a horizontal or a vertical step per time unit. The movement of the prey agent was random. No two agents were allowed to occupy the same location. The goal of this problem was to show the effectiveness of nine organizational structures, with varying degrees of agent cooperation and control, on the efficiency with which the predator agents could capture the prey.

The approach undertaken by Gasser *et al.* (Gasser et al., 1989) allowed for the predators to occupy and maintain a *Lieb configuration* (each predator occupying a different quadrant, where a quadrant is defined by diagonals intersecting at the location of the prey) while homing in on the prey. This study, as well as the study by Singh (Singh, 1990) on using group intentions for agent coordination, lacks any experimental results that allow comparison with other work on this problem.

Stephens and Merx (Stephens and Merx, 1989; Stephens and Merx, 1990) performed a series of experiments to demonstrate the relative effectiveness of three different control strategies. They defined the local control strategy where a predator broadcasts its position to other predators when it occupies a neighboring location to the prey. Other predator agents then concentrate on occupying the other locations neighboring the prey. In the distributed control strategy, the predators broadcast their positions at each step. The predators farther from the prey have priority in choosing their target location from the preys neighboring location. In the centralized–control strategy, a single predator directs the other predators into subregions of the *Lieb configuration*. Stephens and Merx experimented with thirty random initial positions of the predators and prey problem, and discovered that the centralized control mechanism resulted in capture in all configurations. The distributed control mechanism also worked well and was more robust. They also discovered the performance of the local control mechanism was considerably worse. In their research, the predator and prey agents took turns in making their moves. We believe this is not very realistic. A more realistic scenario is for all agents to choose their actions concurrently. This will introduce significant uncertainty and complexity into the problem.

Korf (Korf, 1992) claims in his research that a discretization of the continuous world that allows only horizontal and vertical movements is a poor approximation. He calls this the orthogonal game. Korf developed several greedy solutions to problems where eight predators are allowed to move orthogonally as well as diagonally. He calls this the diagonal game. In Korf's solutions, each agent chooses a step that brings it nearest to the predator. A *max norm* distance metric (maximum of x and y distance between two locations) is used by agents to chose their steps. The predator was captured in each of a thousand random configurations in these games. But the *max norm* metric does not produce stable captures in the orthogonal game; the predators circle the prey, allowing it to escape. Korf replaces the previously used randomly moving prey with a prey that chooses a move that places it at the maximum distance from the nearest predator. Any ties are broken randomly. He claims this addition to the prey movements makes the problem considerably more difficult.

Manela and Campbell investigated the utility of $N \times M$ (predators $\times$ prey) pursuit games as a testbed for DAI research. (Manela and Campbell, 1993) They utilized genetic algorithms to evolve parameters for decision modules. A difference between their domain and the others is that the grid is bounded, and not toroidal, i.e., the neighbors of a cell on the edge are those cells on the other edge. They found that the 4×1 game was not interesting for DAI research. They concluded that $(M + 4) \times M$, $M > 4$, games have the right complexity to be good testbeds. We believe their argument is invalid in our domain where the grid world is toroidal. One benefit of a bounded grid world is that teams of size two and three can effect a capture of the prey by trapping it against the walls. By removing the bounds, the 4×1 game becomes interesting for DAI research.

In our prior research (Haynes et al., 1995b; Haynes et al., 1995a), we have utilized genetic programming to evolve a behavioral strategy to control all of the predator agents in their pursuit of the prey. Each chromosome represented a behavioral strategy which was employed by all of the predator agents. We compared the best strategy evolved by the genetic programming system against our implementations of Korf's algorithms. We found that the evolved strategy was comparable to the hand-crafted ones. Furthermore, we have determined that this problem is deceptively simple: if the agents have no memory and are no allowed communication, there exist simple prey strategies (such as sit still or move in a straight line) which consistently evade capture by the predator strategies.

4 Evolving Coordination Strategies

4.1 Genetic Programming

Holland's work on adaptive systems (Holland, 1975) produced a class of biologically inspired algorithms known as genetic algorithms that can manipulate and develop solutions to optimization, learning, and other types of problems. In order for GAs to be effective, the solution should be represented as n–ary strings (though some recent work has shown that GAs can be adapted to manipulate real–valued features as well). Though GAs are not guaranteed to find optimal solutions (unlike Simulated Annealing algorithms), they still possess some nice provable properties (optimal allocation of trials to substrings, evaluating exponential number of schemas with linear number of string evaluations, etc.), and have been found to be useful in a number of practical applications (Davis, 1991).

Koza's work on Genetic Programming (Koza, 1992) was motivated by the representational constraint in traditional GAs. Koza claims that a large number of apparently dissimilar problems in artificial intelligence, symbolic processing, optimal control, automatic programming, empirical discovery, machine learning, etc. can be reformulated as the search for a computer program that produces the correct input-output mapping in any of these domains. As such, he uses the traditional GA operators for selection and recombination of individuals from a population of structures, and applies them on structures represented in a more expressive language than used in traditional GAs. The representation language used in GPs are computer programs represented as Lisp S–expressions. Although GPs do not possess the nice theoretical properties of traditional GAs, they have attracted a tremendous number of researchers because of the wide range of applicability of this paradigm, and the easily interpretable form of the solutions that are produced by these algorithms (Angeline and Kinnear, Jr., 1996; Kinnear, Jr., 1994; Koza, 1992; Koza, 1994).

A GP algorithm can be described as follows:

1. Randomly generate a population of N programs made up of functions and terminals in the problem.
2. Repeat the following step until termination condition is satisfied:
   (a) Assign fitness to each of the programs in the population by executing them on domain problems and evaluating their performance in solving those problems.
   (b) Create a new generation of programs by applying fitness proportionate selection operation followed by genetic recombination operators as follows:
      - Select N programs with replacement from the current population using a probability distribution over their fitness.
      - Create new population of N programs by pairing up these selected individuals and swapping random sub–parts of the programs.

3. The best program over all generations (for static domains) or the best program at the end of the run (for dynamic domains) is used as the solution produced by the algorithm.

In GP, the user needs to specify all of the functions, variables and constants that can be used as nodes in the S–expression or parse tree. Functions, variables and constants which require no arguments become the leaves of the parse trees and thus are called *terminals*. Functions which require arguments form the branches of the parse trees, and are called *functions* or *non–terminals*. The set of all terminals is called the *terminal set*, and the set of all functions is called the *function set*. In traditional GP, all of the terminal and function set members must be of the same type. Montana (Montana, 1995) introduced STGP, in which the variables, constants, arguments, and returned values can be of any type. The only restriction is that the data type for each element be specified beforehand.

4.2 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. 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.

4.3 Evaluation of Coordination Strategies

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 is used to favor situations where one or more predators surround the prey.

- Finally, 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.

4.4 Encoding of Behavioral Strategies

In Korf's implementation of the predator–prey domain, he utilized the same algorithm to control each of the predator agents. We evolve behavioral strategies to be used by the predator agents. Behavioral strategies are encoded as S–expressions. Terminal and function sets in the pursuit problem are presented in Tables 1 and 2. In our domain, the root node of all parse trees is enforced to be of type `Tack`, which returns the number corresponding to one of the five choices the prey and predators can make (*Here*, *North*, *East*, *West*, and *South*). Notice the required types for each of the terminals, and the required arguments and return types for each function in the function set.

Our choice of sets reflect the simplicity of the solution proposed by Korf. One of our goals is to have a language in which the algorithms employed by Korf can be represented.

Terminal	Type	Purpose
B	Boolean	TRUE or FALSE
Bi	Agent	The current predator.
Pred1	Agent	The first predator.
Pred2	Agent	The second predator.
Pred3	Agent	The third predator.
Pred4	Agent	The fourth predator.
Prey	Agent	The prey.
T	Tack	Random Tack in the range of Here to North to West.

Table 1: Terminal Set

Function	Return	Arguments	Purpose/Return
CellOf	Cell	Agent A and Tack B	Get the cell coord of A in B.
IfThenElse	Type of B and C	Boolean A, Generic B and C	If A then do B else do C. (B and C must have the same type.)
$<$	Boolean	Length A and Length B	If $A < B$, then TRUE else FALSE.
MD	Length	Cell A and Cell B	*Manhattan distance* between A and B.

Table 2: Function Set

5 Establishing 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 way to create teams 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. A similar mechanism as proposed in this paper has been used to successfully co–evolve a set of prototypes for supervised concept classification problems (Knight and Sen, 1995).

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 our conjecture is that it slows down learning. We investigate the utility of allowing multiple programs to participate during the crossover process. We consider the following crossover functions:

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 is the traditional GP crossover mechanism.

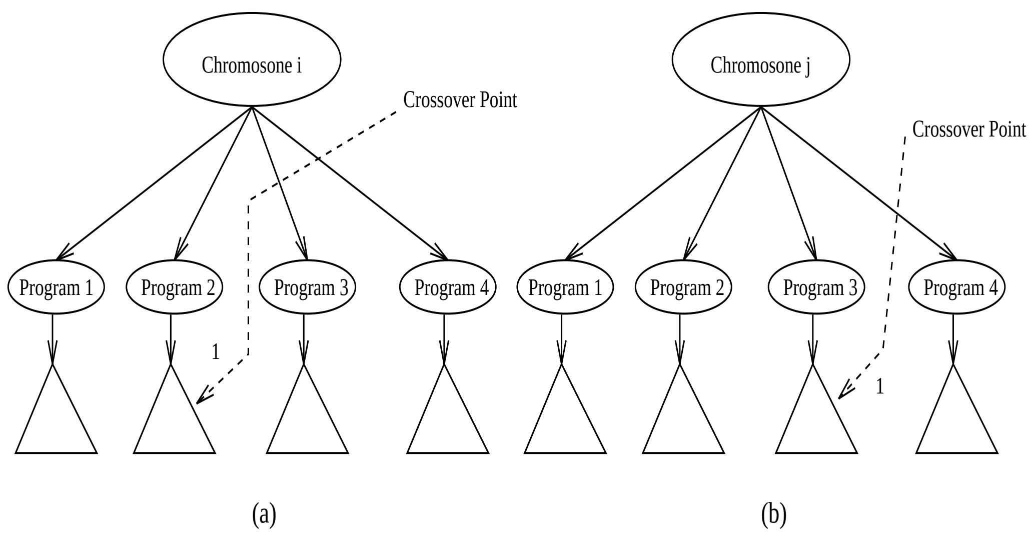

Figure 1: Example crossover for 1 crossover point in a chromosome.

TeamUniform This crossover mechanism is to adapt the uniform crossover function from GA research (Syswerda, 1989) (see Figure 2). Basically we develop a uniform crossover mask for the programs inside a chromosome. A "1" indicates that the programs are copied into the respective child, while a "0" indicates that the programs will 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 can 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 2. The programs have been re–ordered such that $i3$ is paired with $j4$, etc. We utilize this reordering to allow the new crossover method to have the same flexibility as the others, i.e. any branch in one chromosome can engage in crossover with any branch in another chromosome.

Some recent work in competitive co-evolution has involved concurrently evolving agents that compete against each other. Hence individuals from two co-evolving populations can be used to evaluate each other (Rosin and Belew, 1995; Haynes and Sen, 1996; Grefenstette and Daley, 1996). This mode of evolution offers the possibility of a graded variation of environmental challenges which can allow for more effective agents to be developed over time (as opposed to preselecting a set of standard problems for evaluating agents). Work in competitive co-evolution has also included *island models*, which involves evolving subpopulations with occasional migration (Tanese, 1989).

Our team strategies, TeamBranch and TeamUniform, may be thought of as cooperative co-evolution processes where each subpopulation consists of programs that represent one of the agents. There is an exchange of genetic material between two agents from two different subpopulations through the crossover operation; this exchange of information may be likened to migration between the subpopulations. In our implementation, we dictate that the k-th team or structure is formed by the k-th member of each subpopulation. The evaluation of a team is shared by all the members of the team. In the work of Potter *et al.*, a team is formed by combining a member of the current subpopulation with the best members received from the other subpopulations (Potter et al., 1995). The individual member of the subpopulation then receives an evaluation corresponding to the performance of the group thus formed. In our work, we can view each team member to be receiving the same evaluation as the entire team. Our assumption of allowing sharing of genetic information between team members is also used by other GA researchers working on the problem of cooperative co-evolution (Bull and Fogarty, 1996).

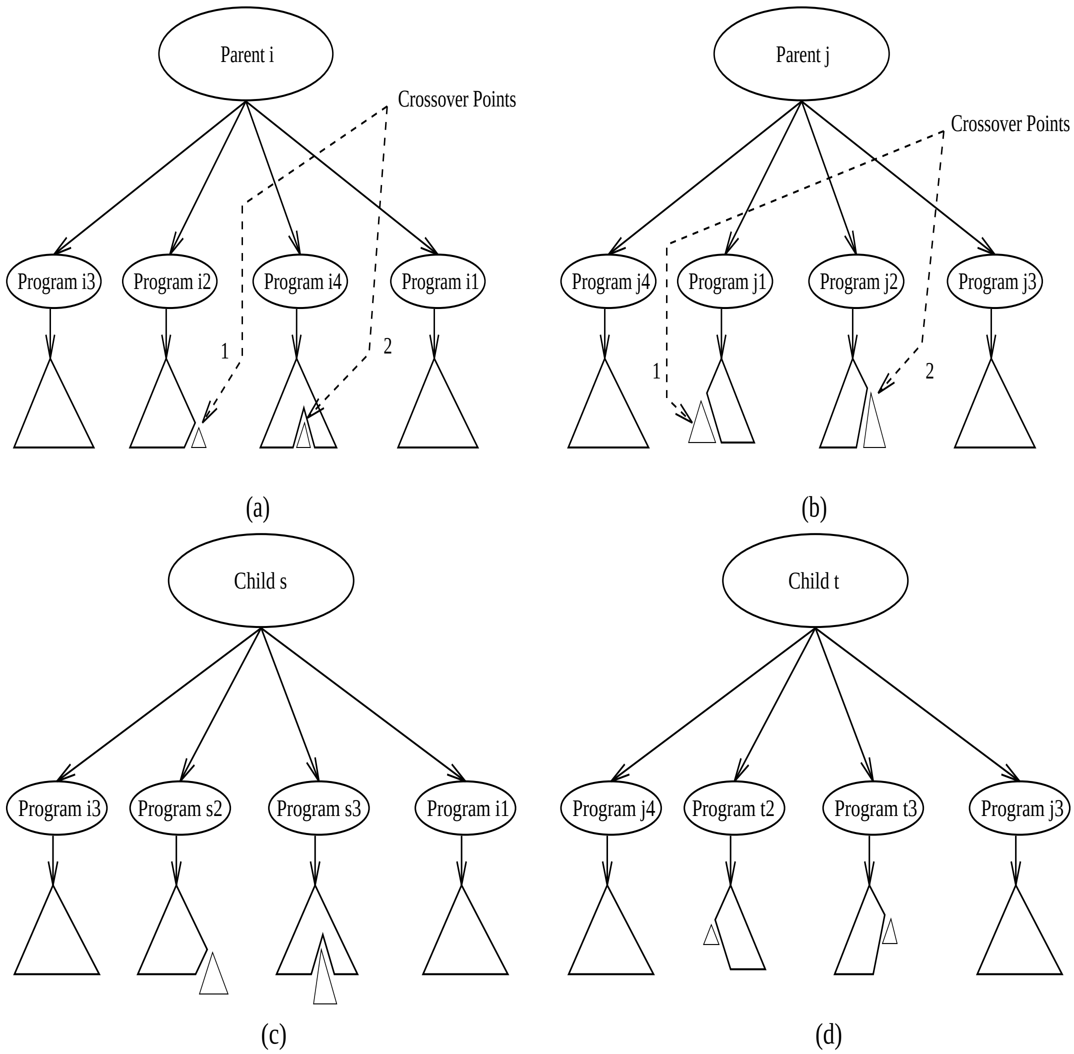

Figure 2: Example uniform crossover for the mask {1001}. (a) has Parent i with an ordering of (3241). (b) has Parent j with an ordering of (4123). (c) has Child s, with two branches created via crossover. (d) has Child t, with two branches created via crossover.

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 are as follows: a population size of 600, a maximum of 1000 generations, a crossover rate of 90% and a mutation rate of 10% (we employ the standard mutation operator utilized in GP, i.e., if a chromosome is selected for mutation, than randomly pick a node and replace its subtree with a randomly generated subtree). In each generation, each chromosome is evaluated for the same three random initial placements of predators and prey. We ran each approach with the same six different seeds for the random number generator. The averaged results for the best fitness per generation for the three crossover functions are shown in Figure 3.

While TeamBranch initially learns faster than TeamTree, TeamTree is able to learn to the same degree of cooperation. The primary significant trend that we observe from our experiments is that while TeamBranch initially learns faster than TeamUniform, TeamUniform is able to noticeably outperform TeamBranch in the long run. This shows that it can be more effective to evolve a set of possibly heterogeneous agents rather than using a homogeneous agent group.

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 (Haynes et al., 1996). 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 the predator stuck behind the other one cannot get to a capture position. With the heterogeneous behavioral strategies, deadlock situations have the potential to be avoided.

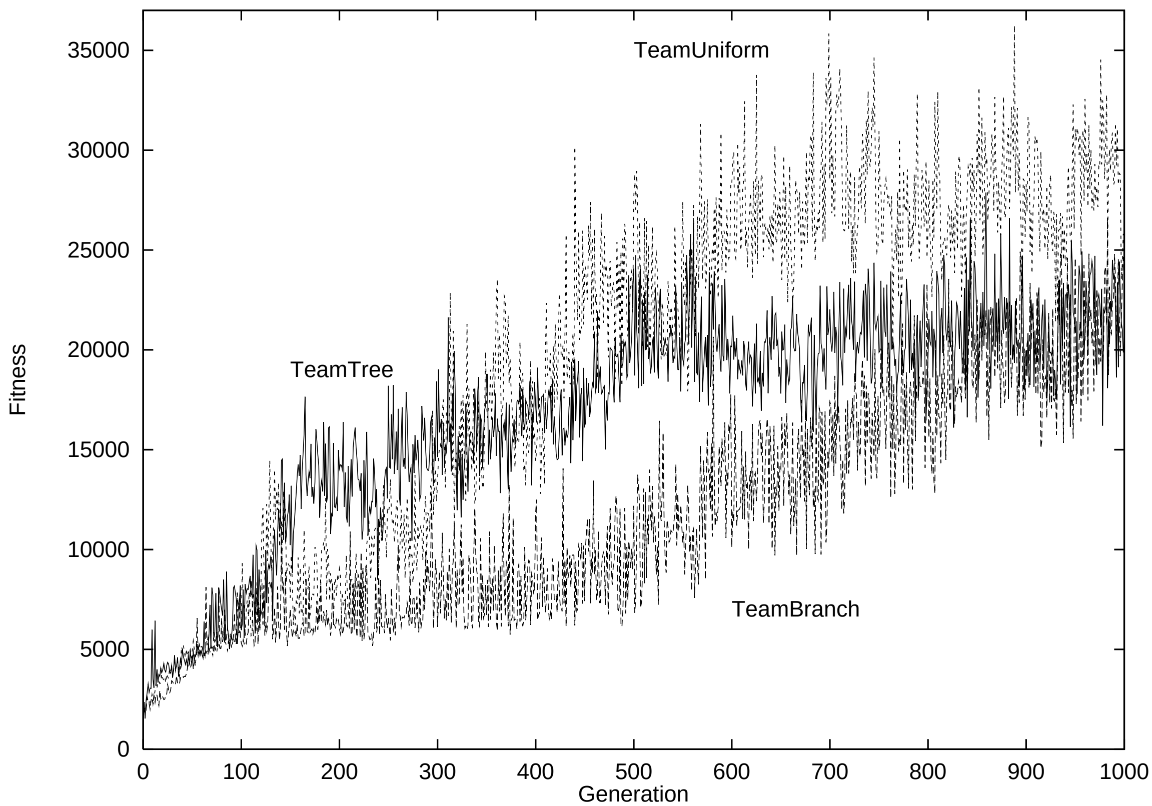

Figure 3: Best Fitness for the crossover functions. (maximum possible fitness is 48,000)

7 Conclusions

Computational agent societies are rapidly approaching. As human users come to expect more capabilities from computational resources, more effort will be spent in developing agent groups that can cooperate to meet a variety of user needs. Whereas agent developers can invest time in developing more extensive and effective agent capabilities, agents from different developers must learn to reduce conflict for resources via cooperation. Due to the nature of open environments, such learning is best done on-line, through interaction with other agents.

We have presented a learning strategy that improves coordination through an evolutionary algorithmic approach. The heterogeneous agents are able to improve co-adaption via the introduction of a new genetic programming crossover function, TeamUniform. Instead of limiting crossover to one result producing branch, multiple result producing branches are capable of exchanging genetic material.

We have also found that heterogeneous agents have been better able to excel compared to homogeneous agent groups in a symmetrical domain. It would seem likely that heterogeneous agents would suffer from the lack of simple models of others (a capability which can be exploited 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 still perform different actions. This asymmetry of behavioral strategies allows the agents to avoid potential deadlock situations.

8 Future Work

The predator–prey domain is very symmetrical, which favors the emergence of homogeneous agents. To explore the degree of similarity of different behavioral strategies in a group, we need to develop some tools to enable us to analyze the similarity of two chromosomes; both in semantical and syntactical content. This is evidenced by there being two team members with different subtrees, but with identical results.

We also want to explore the emergence of co–operative evolution in the context of domains which are asymmetrical. We believe that in this type of scenario, specialists will develop. In the context of the animal hunting example, we would expect one scout, one flusher, and one attacker to develop. Role assignments can be static or be adaptive to take advantage of varying environmental conditions.

We envision evolving a team consisting of generalists and specialists. Some critical tasks are always handled by the specialists, while generalists switch back and forth between tasks as environmental demands vary. In a computer network, any machine can host user sessions. But software, such as a compiler or word processor, might be restricted to certain machines due to licensing agreements. Also, certain machines in the network might have special hardware, such as printers, modems, plotters, etc. A process scheduler and loader could be evolved to evenly spread tasks amongst the machines.

We are also interested in evolutionary mechanisms that allow quick adaptation to environmental changes. The use of responsive adaptation mechanisms would allow us to evolve agent groups and modify them as the environmental demands vary. Such adaptation schemes could also include the addition of new members to the team, and as a consequence a reorganization of the team that will enable the most effective utilization of their resources.

References

Angeline, P. and Kinnear, Jr., K. E., editors (1996). *Advances in Genetic Programming 2.* MIT Press, Cambridge, MA, USA.

Angeline, P. J. and Pollack, J. B. (1993). 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.

Barbuceanu, M. and Fox, M. S. (1995). COOL: A language for describing coordination in multiagent systems. In Lesser, V., editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 17–24, San Francisco, CA. MIT Press.

Benda, M., Jagannathan, V., and Dodhiawala, R. (1986). On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington.

Bond, A. H. and Gasser, L., editors (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann.

Bull, L. and Fogarty, T. C. (1996). Evolution in cooperative multiagent environments. In Sen, S., editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 22–27, Stanford University, CA.

Davis, L., editor (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York, NY.

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

Finin, T., Fritzon, R., McKay, D., and McEntire, R. (1994). KQML – A language and protocol for knowledge and information exchange. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence*, pages 126–136, Seatle, WA.

Fox, M. S. (1981). An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 140–150, Morgan Kaufmann, 1988.).

Gasser, L., Rouquette, N., Hill, R. W., and Lieb, J. (1989). Representing and using organizational knowledge in DAI systems. In Gasser, L. and Huhns, M. N., editors, *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*, pages 55–78. Pitman.

Gotwald, Jr., W. H. (1995). *Army Ants: The Biology of Social Predation*. Cornell University.

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

Grefenstette, J. and Daley, R. (1996). Methods for competitive and cooperative co-evolution. In Sen, S., editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 45–50, Stanford University, CA.

Haynes, T., Lau, K., and Sen, S. (1996). Learning cases to compliment rules for conflict resolution in multiagent systems. In Sen, S., editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 51–56, Stanford University, CA.

Haynes, T. and Sen, S. (1996). Evolving behavioral strategies in predators and prey. In Weiß, G. and Sen, S., editors, *Adaptation and Learning in Multi–Agent Systems*, Lecture Notes in Artificial Intelligence, pages 113–126. Springer Verlag, Berlin.

Haynes, T., Sen, S., Schoenefeld, D., and Wainwright, R. (1995a). Evolving multiagent coordination strategies with genetic programming. Technical Report UTULSA-MCS-95-04, The University of Tulsa.

Haynes, T., Wainwright, R., Sen, S., and Schoenefeld, D. (1995b). Strongly typed genetic programming in evolving cooperation strategies. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 271–278, San Francisco, CA. Morgan Kaufmann Publishers, Inc.

Holland, J. H. (1975). *Adpatation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

Kinnear, Jr., K. E., editor (1994). *Advances in Genetic Programming.* MIT Press, Cambridge, MA.

Knight, L. and Sen, S. (1995). PLEASE: A prototype learning system using genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 429–435. Morgan Kaufmann Publishers, Inc.

Korf, R. E. (1992). A simple solution to pursuit games. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 183–194.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT Press, Cambridge, MA.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA.

Krebs, J. R. and Davies, N. B. (1993). *An Introduction to Behavioural Ecology.* Blackwell Scientific Publications.

Lesser, V. R. (1995). Multiagent systems: An emerging subdiscipline of AI. *ACM Computing Surveys*, 27(3):340–342.

Levy, R. and Rosenschein, J. S. (1992). A game theoretic approach to the pursuit problem. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 195–213.

Malone, T. W. (1987). Modeling coordination in organizations and markets. *Management Science*, 33(10):1317–1332. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 151–158, Morgan Kaufmann, 1988.).

Malone, T. W. (1990). Organizing information processing systems: Parallels between human organizations and computer systems. In Robertson, S. P., Zachary, W., and Black, J. B., editors, *Cognition, Computing, and Cooperation*, pages 56–83. Ablex Publishing Corporation.

Manela, M. and Campbell, J. A. (1993). Designing good pursuit problems as testbeds for Distributed AI: a novel application of Genetic Algorithms. In *Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Neuchâtel, Switzerland.

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

Mullen, T. and Wellman, M. P. (1995). A simple computational market for network information services. In Lesser, V., editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 283–289, San Francisco, CA. MIT Press.

Potter, M. A., Jong, K. A. D., and Grefenstette, J. J. (1995). A coevolutionary approach to learning sequential decision rules. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 366–372. Morgan Kaufmann Publishers, Inc.

Reynolds, C. W. (1994). Competition, coevolution and the game of tag. In *Artificial Life IV*. MIT Press.

Robbins, S. P. (1993). *Organizational Behavior: Concepts, Controversies, and Applications*. Prentice Hall.

Rosin, C. D. and Belew, R. K. (1995). Methods for competitive co–evolution: Finding opponents worth beating. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 373–380. Morgan Kaufmann Publishers, Inc.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Sen, S. (1996). Adaptation, coevolution and learning in multiagent systems. Technical Report SS-96-01, AAAI Press, Stanford, CA.

Shaw, M. J. (1996). Cooperative problem solving and learning in multi-agent information systems. *International Journal of Computational Intelligence and Organizations*.

Singh, M. P. (1990). The effect of agent control strategy on the performance of a DAI pursuit problem. In *Working Papers of the 10th International Workshop on Distributed Artificial Intelligence*.

Stephens, L. M. and Merx, M. B. (1989). Agent organization as an effector of DAI system performance. In *Working Papers of the 9th International Workshop on Distributed Artificial Intelligence*.

Stephens, L. M. and Merx, M. B. (1990). The effect of agent control strategy on the performance of a DAI pursuit problem. In *Proceedings of the 1990 Distributed AI Workshop*.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann.

Tanese, R. (1989). Distributed genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–440, San Mateo, CA. Morgan Kaufman.

Weiß, G. and Sen, S., editors (1996). *Adaptation and Learning in Multi–Agent Systems*. Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin.