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

Abstract

We introduce a cooperative co–evolutionary system to facilitate the development of teams of agents. Specifically, we deal with the credit assignment problem of how to fairly split the fitness of a team to all of its participants. We believe that k different strategies for controlling the actions of a group of k agents can combine to form a cooperation strategy which efficiently results in attaining a global goal. A concern is the amount of time needed to either evolve a good team or reach convergence. We present several crossover mechanisms to reduce this time. Even with this mechanisms, the time is large; which precluded the gathering of sufficient data for a statistical base.

1 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) [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. 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 [Haynes *et al.*, 1995], 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.

Our approach for developing coordination strategies for multi–agent problems is completely domain independent, and uses the strongly typed genetic programming (STGP) paradigm [Montana, 1994], which is an extension of genetic programming (GP) [Koza, 1992]. 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 [Benda *et al.*, 1985] 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; 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.

In this work we examine the rise of cooperation strate-

gies without implicit communication. In our previous research, the developed strategies had implicit communication in that the same program was used to control the four 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 cod-evolution problem as described in [Angeline and Pollack, 1993; Haynes and Sen, 1995; Reynolds, 1994]. We believe that cooperative cod-evolution provides opportunities to produce solutions to problems that cannot be solved with implicit communication.

The rest of this paper is laid out as follows: Section 2 introduces the pursuit domain, and describes the experimental setup. Section 3 presents three crossover strategies, beyond simple point based crossover, for improving the learning of the team. Section 4 shows the relevance of this work to current research topics in GP. Section 5 compares the utility of several of the crossover strategies as they evolve teams. Section 6 wraps up our research into team formation. Section 7 points out how this work can be extended.

2 Pursuit Domain

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 move, 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'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. Furthermore, two predators cannot communicate to resolve conflicts or negotiate a capture strategy. These two rules eliminate explicit communication between agents.

2.1 Evaluation of Coordination Strategies for Predators

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 STGPs 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.

3 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 [Grefenstette, 1988]. 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 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].

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

3.1 TeamBranch

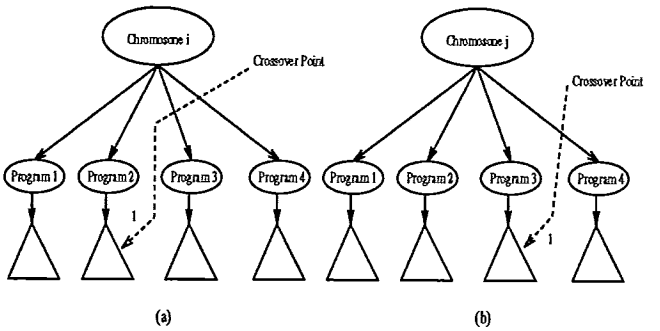

Figure 1: Example crossover for 1 crossover point in a chromosome.

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.

3.2 TeamAll

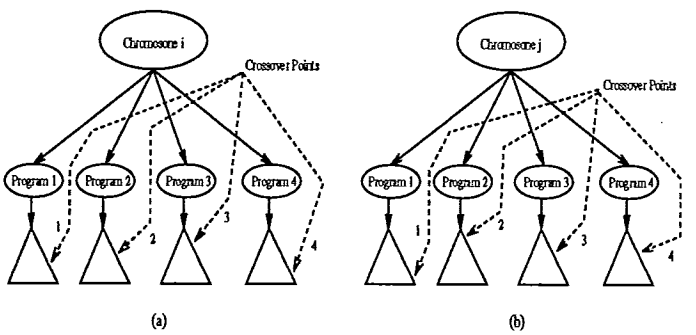

Figure 2: Example crossover for all programs in a tree. A crossover point is selected in the subtree of each program. Thus there are four crossovers taking place; between each program P_i for the two chromosomes.

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 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. This is different than recombination in that there is only one "parent" and one resultant "child".

3.3 TeamUniform

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 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.

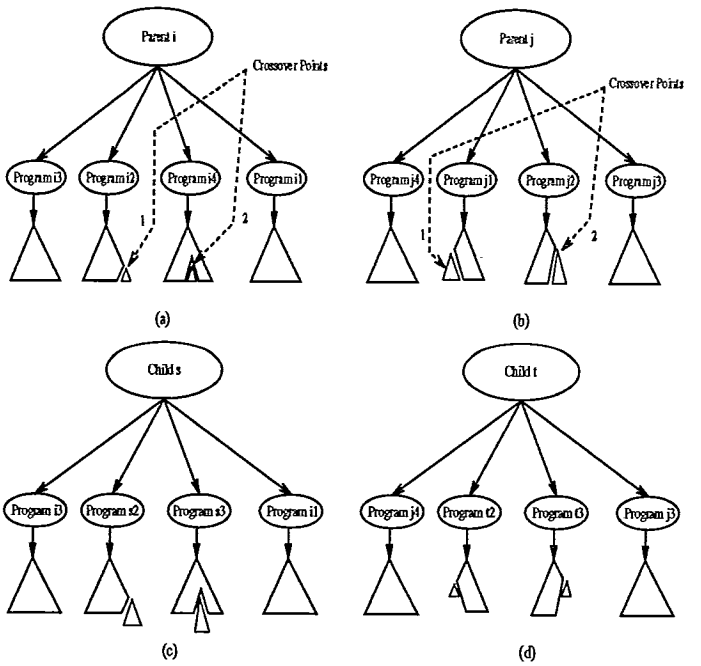

Figure 3: Example uniform crossover for the mask (1001). (a) has Parent i with an ordering of (3241). (b) has Parent j with an ordering of (4123). (c) has Child s, with two children created via crossover. (d) has Child t, with two children created via crossover.

3.4 TeamKCross

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.

4 Implications of Research

There has been previous research into S–expressions containing more than one executable branch. Both Andre [Andre, 1995] and Haynes [Haynes, 1994] have investigated systems in which one branch of the S–expression manipulate a memory structure and the other branch utilizes the memory structure to interact with an environment. Andre explicitly creates two programs in the S–expression, while Haynes relies on strong typing to force the root node to develop two branches for the construction and utilization of memory. Both of these systems can be considered to utilize the Pitt approach to credit assignment.

While Koza's Automatically Defined Functions (ADF) [Koza, 1994] are not separate "agents", they utilize many different branches, say k, to facilitate learning. In the GP mailing list, Siegel [Siegel, 1994] posed the question as to whether some form of crossover utilizing $k > 1$ would help in the learning process? The replies were mixed, and pointed out the need for further research.

5 Results

We have tested two different approaches to constructing teams for the predator agents: TeamBranch, Section 3.1, each team member has its own program, i.e. subtree,

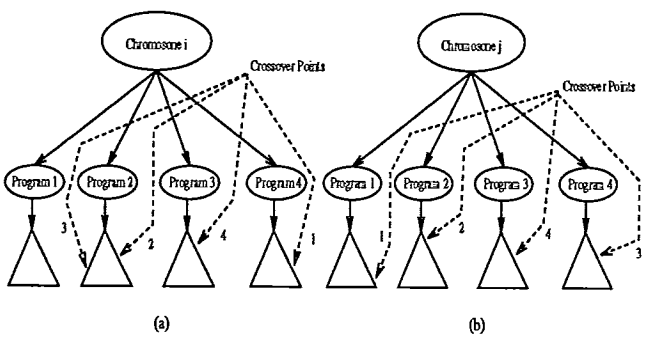

Figure 4: Example crossover k crossover points in a chromosome.

and there is one branch crossover. TeamAll, Section 3.2, each team member has its own program, i.e. subtree, and there is $k = 4$ branch crossover. Finally, we compare the above strategies against the method we have utilized in our previous research: TeamTree, each team member shares the same program.

The basic setup for each experiment was a population size of 600, a maximum of 1000 generations, and a maximum fitness of 48,000. Note that even if either a good solution or convergence was found before 1000 generations, we let the system continue. In order to be somewhat fair, we ran each approach with the same four different initial seeds for the random number generator. We would have liked to have a better statistical base, but each run takes between four to ten days, depending on the Sun system used.

5.1 Population Results

The averaged results for the Best and Average Fitnesses per generation are shown in Figures 5 (TeamTree), 6 (TeamBranch), and 7 (TeamAll). In general the Best of Generation curves have reached a plateau of about 20,000 to 22,000 fitness points. If we look at the "center" of the plateaus, then the systems can be ranked in the order of TeamTree, TeamAll, and TeamBranch. These curves can also be broken into three regions, although it is harder to do so for TeamBranch, Figure 6.

The differences occur in reaching that plateau. The TeamTree curve learns faster than both TeamBranch and TeamAll. It reaches both of its plateaus before the other systems. The TeamAll system learns faster than the TeamBranch. Notice that TeamAll reaches the plateau level about 200 generations before TeamBranch.

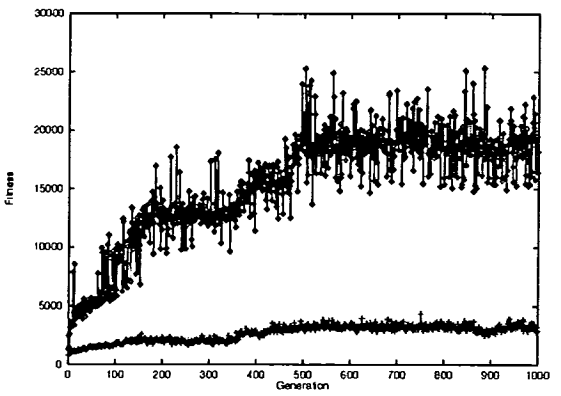

Figure 5: Average and Best Fitness for no branches.

5.2 Team Analysis

We have also tested the best individuals per system for both 1000 random test cases and the 30 standard test cases as set forth by Stephens and Merx [Stephens and Merx, 1989]. Each team was allowed to move for 200 time steps. None of the teams did very well.

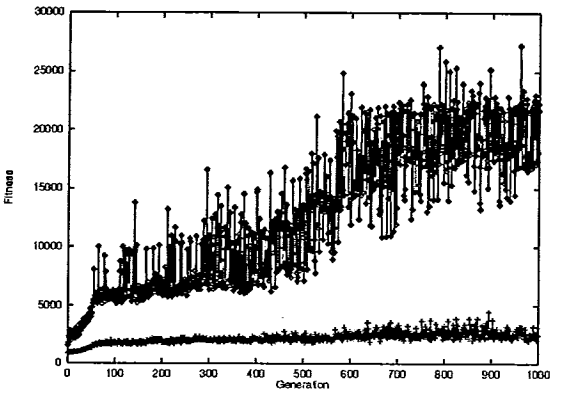

Figure 6: Average and Best Fitness for 1 branch crossover.

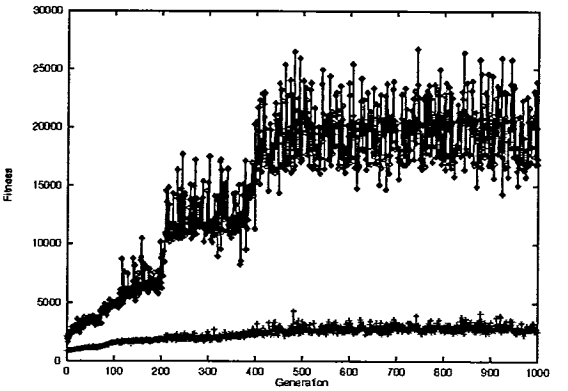

Figure 7: Average and Best Fitness for all crossover.

In Table 1, we present the results of the prey having a move away from nearest predator (MAFNP) algorithm. Due to space considerations, we do not present other prey algorithms. We do include performance results of agents controlled by the MN, MNO, MD, and MDO for comparison. The STGP algorithm is the evolved strategy reported in [Haynes *et al.*, 1995], and the other algorithms are based on the *max norm* (MN and MNO) and *Manhattan distance* (MD and MDO) algorithms. They are discussed in detail in [Haynes *et al.*, 1995]. The STGP strategy represents the TeamTree team.

The TeamAll team did better than the TeamBranch team, but they both did worse than the STGP algorithm. Also, 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. STGP.

The moves taken by the STGP program (strategy) for various relative positions of a predator with respect to the prey are graphically represented in Figure 8. In Figure 9, the moves taken by the four predators, from the TeamAll strategy A1, are graphically represented. Note that Figure 9(a) and Figure 9(b) are very simi-

	Prey First			Prey Sync		
	Captures	Steps	Blocks	Captures	Steps	Blocks
STGP	0.385(0.496) 0.385(0.496)	29.500(24.185) 29.500(24.185)	5.846(2.412)	14.846(1.642) 14.846(1.642)	109.412(11.872) 109.412(11.872)	3.462(1.529)
A1	10.308(1.995) 10.308(1.995)	115.466(13.152) 115.466(13.152)	2.462(1.794)	0.115(0.326) 0.115(0.326)	99.333(98.241) 99.333(98.241)	0.577(0.809)
B1	6.000(1.811) 6.000(1.811)	107.000(16.859) 107.000(16.859)	1.692(1.123)	0.962(0.720) 0.962(0.720)	109.160(70.818) 109.160(70.818)	0.654(0.745)
MNO	0.000(0.000) 0.346(0.562)	0.000(0.000) 104.333(94.001)	0.346(0.562)	0.000(0.000) 0.308(0.471)	0.000(0.000) 99.875(89.788)	0.385(0.571)
MDO	2.423(1.528) 2.423(1.528)	56.254(34.139) 56.254(34.139)	13.731(3.672)	2.923(1.742) 2.923(1.742)	55.263(23.194) 55.263(23.194)	14.077(2.576)
MN	0.077(0.272) 2.000(1.265)	199.000(194.979) 112.615(53.095)	1.731(1.589)	0.000(0.000) 0.538(0.647)	0.000(0.000) 127.071(100.675)	1.385(1.098)
MD	15.808(2.173) 15.808(2.173)	101.182(12.816) 101.182(12.816)	3.231(1.394)	17.615(2.654) 17.615(2.654)	108.919(11.340) 108.919(11.340)	2.000(1.296)

Table 1: Average number of captures for MAFNP Prey (standard deviations are presented in parentheses).

lar movement strategies. This observation suggests that the predator agents are learning the same behavioral strategy, 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.

6 Conclusions

We believe that the TeamAll strategy for building a team is better than the TeamBranch strategy. In particular, with each agent's program engaging in crossover versus only one of the agent's programs engaging in crossover, learning is faster. The TeamTree building strategy fares better than either of the other two. We believe that this feature is due to the implicit communication that the TeamTree team members have available. It is our conjecture that the others can be evolving implicit communication in that team members are developing similar behavioral strategies.

Our belief that the TeamAll strategy is better than the TeamBranch strategy has been supported by the collected data. As we gather further data, we will see if this is truly statistically significant.

7 Future Work

Clearly we need to run more test cases in order to build a statistical basis for significance testing. We also need to experiment with the uniform crossover and k–point crossover mechanisms mentioned earlier.

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.

8 Acknowledgments

This research was partially supported by OCAST Grant AR2–004, NSF Research Initiative Award IRI–9410180 and Sun Microsystems, Inc.

We would also like to thank the anonymous reviewers for many helpful comments, in particular for the reference to the ADF discussion in the GP mailing list. Una–May O'Reilly was very helpful in forwarding a copy of the 1994 mailing list.

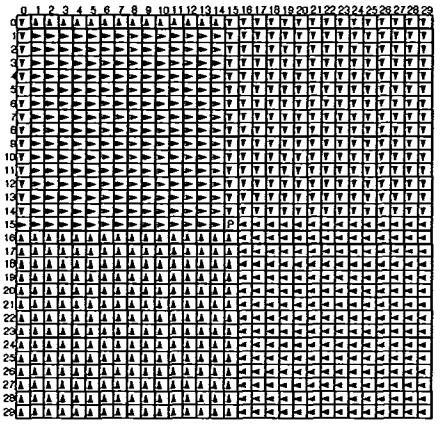

Figure 8: Pursuit path found by STGP.

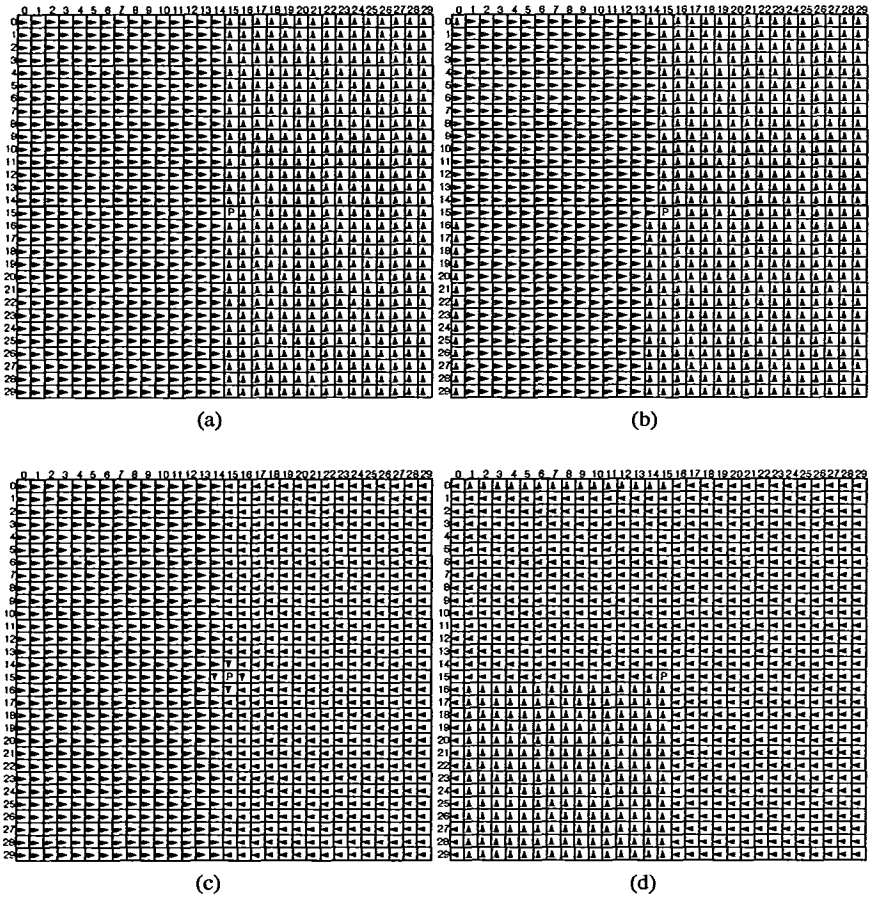

Figure 9: Pursuit paths found by A1.

References

[Andre, 1995] David Andre. The evolution of agents that build mental models and create simple plans using genetic porgramming. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 248–255, 1995.

[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.

[Benda *et al.*, 1985] 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.

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

[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.

[Gasser *et al.*, 1989] 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.

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

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

[Haynes *et al.*, 1995] 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*, pages 271–278, 1995.

[Haynes, 1994] Thomas D. Haynes. A simulation of adaptive agents in a hostile environment. Master's thesis, University of Tulsa, Tulsa, OK., April 1994.

[Knight and Sen, 1995] Leslie Knight and Sandip Sen. Please: A prototype learning system using genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 429–435, 1995.

[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 Means of Natural Selection.* MIT Press, 1992.

[Koza, 1994] John R. Koza. *Genetic Programming II, Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[Levy and Rosenschein, 1992] 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.

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

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

[Siegel, 1994] Eric Siegel. ADF Crossover – response summary. Genetic Programming Mailing List, June 15th 1994.

[Stephens and Merx, 1989] 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.

[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.