Evolving Cooperation Strategies *

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

Abstract

The identification, design, and implementation of strategies for cooperation is a central research issue in the field of Distributed Artificial Intelligence (DAI). We propose a novel approach to the construction of cooperation strategies for a group of problem solvers based on the Genetic Programming (GP) paradigm. GPs are a class of adaptive algorithms used to evolve solution structures that optimize a given evaluation criterion. Our approach is based on designing a representation for cooperation strategies that can be manipulated by GPs. We present results from experiments in the predator-prey domain, which has been extensively studied as a easy-to-describe but difficult-to-solve cooperation problem domain. The key aspect of our approach is the minimal reliance on domain knowledge and human intervention in the construction of good cooperation strategies. Promising comparison results with prior systems lend credence to the viability of this approach.

Topic areas: Evolutionary computation, cooperation strategies

1 Introduction

Researchers in the field of DAI have invested considerable time and effort in identifying domains where multiple, autonomous agents share goals and resources, and need to use mutually acceptable work-sharing strategies to accomplish common goals. Developing cooperation strategies to share the work load is a daunting problem when the environment in which the agents are working is incompletely understood, and/or is uncertain. Current approaches to developing cooperation strategies are mostly off-line mechanisms, that use extensive domain knowledge to design from scratch the most appropriate cooperation strategy (in most cases a cooperation strategy is chosen if it is reasonable, as it is impossible to prove the existence of and identify the best cooperation strategy).

We propose a new approach to developing cooperation strategies for multi-agent problem solving situations. Our approach is different from most of the existing techniques for constructing cooperation strategies in two ways:

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

The approach proposed in this paper is completely domain independent, and uses the GP paradigm to develop, through repeated problem solving, increasingly efficient cooperation strategies. GPs use populations of structures that are evaluated by some domain-specific evaluation criterion. The structures are stored as Lisp symbolic expressions (S–expressions) and are manipulated such that better and better structures are evolved by propagating and combining parts of structures that fare well (as measured by the evaluation criterion) compared with other structures in the population. GPs are a recent offshoot of genetic algorithms,

*Research partially supported by OCAST Grant AR2-004 and Sun Microsystems, Inc.

and share the same biological motivations in propagating more fit structures, but use a richer representation language. To use the GP approach for evolving cooperation strategies, we have to find an encoding of strategies as S-expressions and choose an evaluation criterion for a strategy corresponding to an arbitrary S-expression. The mapping of strategies to S-expressions and vice versa can be done by a set of functions and literals representing the primitive actions in the domain of application. Evaluations of structures or more appropriately, the strategies represented by the structures, can be done by allowing the agents to execute the particular strategies in the application domain and measure their efficiency and effectiveness by any set of criteria relevant to the domain.

To test our hypothesis that useful cooperation strategies can be thus evolved for non-trivial problems, we decided to use the predator-prey pursuit game [1], a domain often used to test out new approaches to developing cooperation schemes. A wide variety of approaches [3, 7, 10, 15, 16, 17] have been used to study this domain where multiple predator agents try to capture a prey agent by surrounding it.

The rest of the paper is organized as follows: Section 2 describes the pursuit game in more detail and presents brief summaries of results from some of the prior approaches to solving this problem; Section 3 presents a brief introduction to the GP paradigm, followed by a discussion of our encoding of cooperation strategies in the pursuit problem in the form of S-expressions, which can be manipulated by the GP method; Section 4 presents results from our experiments on evolving cooperation strategies with GP in the predator-prey pursuit domain, and Section 5 presents our conjectures on the applicability of our approach to developing cooperation strategies in different problem domains.

2 The pursuit problem

The Predator-Prey pursuit problem is a common domain used in Distributed Artificial Intelligence (DAI) research to evaluate techniques for developing cooperation strategies. The original version of this problem was introduced by Benda, *et al.* [1] and consisted of four blue (predator) agents trying to capture a red (prey) agent by surrounding it from four directions on a gridworld. Agent movements were limited to one horizontal or vertical step per time unit. The movement of the red agent was random (chose a neighboring location, not occupied by a blue agent, randomly), and multiple predators were allowed to occupy the same location. The goal of this work was to show the effectiveness of nine organizational structures, with varying degrees of agent cooperation and control, on the efficiency with which the blue agents could capture the red agent.

The approach undertaken by Gasser *et al.* [3] was 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 [15] on using group intentions for agent coordination, lacks any experimental results that allow comparison with other work on this problem.

Stephens and Merx [16, 17] performed a series of experiments to demonstrate the relative effectiveness of different control strategies (local control: a predator broadcasts its position to others when it occupies a neighboring location to the prey, others then concentrate on occupying the other locations neighboring to the prey; distributed control: predators broadcast their positions at each step, and those further off get to choose their target location from the preys neighboring location; centralized-control: a single predator directs other predators into subregions of the *Lieb configuration* mentioned above). They experimented with thirty random initial positions of the predators and prey and found that the centralized control mechanism resulted in capture in all configurations. The distributed control mechanism also worked well and was more robust, whereas the performance of the local control mechanism was considerably worse. We believe that the reason for their high success rate was that the predator and prey agents took turns in making their moves. A more realistic scenario would be for all agents to choose their actions concurrently, which will introduce significant uncertainty and complexity into the problem.

Levy and Rosenschein [10] use results from game theory on cooperative and non-cooperative games to choose optimal moves for the predators. Whereas their method minimizes communication between agents, it is computationally intensive, and does not provide a comparable capture rate. It also assumes that each predator can see the locations of all other predators.

Finally, Korf [7] claims that a discretization of the continuous world that allows only horizontal and vertical movements (he calls this the orthogonal game) is a poor approximation, and provides greedy solutions to problems where eight predators are allowed to move diagonally as well (the diagonal game), and a world in which six predators move on a hexagonal rather than rectangular grid (the hexagonal game). In his design, 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 to solve all the three games; the predator was captured in each of thousand random configurations in these games. He admits that the *max norm* distance metric, though suitable for the diagonal and the hexagonal game, is difficult to justify for the orthogonal game. To improve the efficiency of capture (steps taken for capture), he adds a term to the evaluation of moves that enforces predators to move away from each other (and hence encircle the prey) before converging on the prey (thus eliminating escape routes). This measure succeeds admirably in the diagonal and hexagonal games but makes the orthogonal game unsolvable. Korf replaces a randomly moving prey with a prey that chooses a move which puts it at the maximum distance from the nearest predator (ties are still broken randomly). He claims this makes the problem considerably more difficult, but we believe that is the case only if predators and prey take turns for moving (as in his experiments).

We use a 30 by 30 grid with the initial configuration consisting of the prey in the center and the predators placed in random non-overlapping positions (a smaller grid is shown in Figure 1). In our experiments, all agents choose their action simultaneously, the world is accordingly updated (may need some conflict resolution), and the agents choose their action again based on the updated world state. 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 moves away from the nearest predator, but 10% of its choices are to stay where it is (this effectively makes the predators travel at a greater speed compared to the prey). The grid is toroidal in shape, and the games are of the orthogonal form. A predator can see the prey, but not other predators; neither do they possess any explicit communication skills (that is, two predators cannot communicate to resolve conflicts or negotiate a capture strategy).

3 Genetic Programming

Holland's work on adaptive systems [5] produced a class of biologically inspired algorithms known as genetic algorithms (GAs) 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 [2].

Koza's work on Genetic Programming [8] 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, in a short period of time 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 [6, 9].

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 fitnesses 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 fitnesses.
   - 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

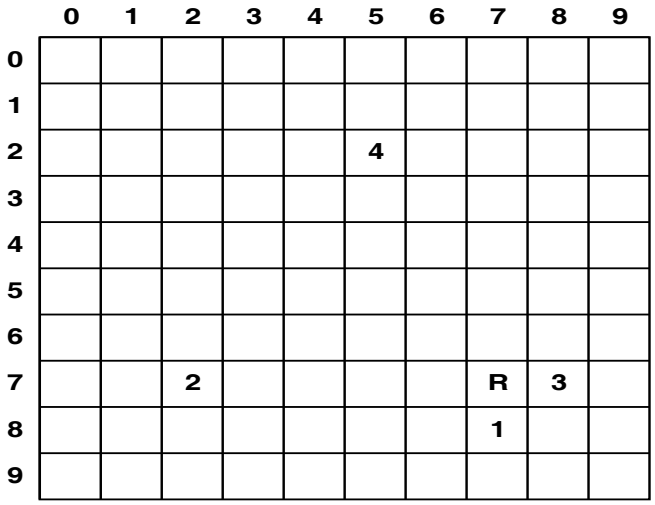

Figure 1: Example of a 10 by 10 grid (R is the prey, the predators are numbered 1-4).

run (for dynamic domains) is used as the solution produced by the algorithm.

For our experiments, a GP algorithm was put to the task of evolving a program that is used by a predator to choose its moves. The same program was used by all the predators. Each program in the population, therefore, represented a strategy for implicit cooperation to capture the prey. We postpone the discussion of evolution of these programs to Section 4.

Terminal	Type	Purpose
B	Boolean	TRUE or FALSE
Bi	Agent	The current predator.
Prey	Agent	The prey.
T	Tack	Random Tack in the range of Here to North to West.

Table 1: Terminal Set

3.1 Encoding of cooperation strategies

GP generated programs are S–expressions which can be represented by the corresponding parse trees. The leaf nodes of such trees are occupied by an element of the terminal set, and other nodes are occupied by elements of the function set. Terminal and function sets are determined by the domain of application; the choice of these sets in the pursuit problem are presented in Table 1 and Table 2. In our domain, the root node of all parse trees are enforced to be of type *Tack*, which returns the number corresponding to one of the five choices an agent, i.e. a predator, can make (*North*, *East*, *West*, *South* and *Here*).

3.2 Evaluation of cooperation strategies

To evolve cooperation strategies using GPs we need to rate the effectiveness of cooperation strategies represented as programs or S–expressions. We chose to evaluate such strategies by putting it to task on k randomly generated pursuit scenarios. On each scenario, a program is run for 100 time steps (moves made by each agent), which comprises one simulation. Percentage of capture seems like a good measure of fitness when we are comparing several strategies. But, since the initial population of strategies are randomly generated, and hence it is very unlikely that any of these strategies will produce a capture, we need additional terms in the fitness function to differentially evaluate these "bad" strategies. The key aspect of GPs or GAs is that even though a particular structure is not effective, it may contain useful sub-parts which when combined with other useful subparts, 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 to contain the following terms:

- After each move made according to the strategy, for each predator a fitness of $\frac{\text{Grid width}}{\text{Distance of predator from prey}}$ is added to the fitness of the program representing the strategy. The closer the strategy brings the predators to the

Function	Return	Arguments	Purpose/Return
CellOf	Cell	Agent A and Tack B	Get the cell # 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, the TRUE else FALSE.
MD	Length	Cell A and Cell B	Manhattan distance between A and B.

Table 2: Function Set

```
IFTE( <( IFTE( T,
               MD( CellOf( Prey, H ),
                   CellOf( Bi,E )),
               MD( CellOf( Prey, N),
                   CellOf( Bi, H ))),
        MD( CellOf( Prey, N ), CellOf( Bi, W ))),
      IFTE( <( MD( CellOf( Bi, N),
                   CellOf( Prey, H )),
               MD( CellOf( Bi, N),
                   CellOf( Prey, N ))),
            N,
            E ),
      IFTE( <( MD( CellOf( Prey, N),
                   CellOf( Bi, N)),
               MD( CellOf( Bi, E),
                   CellOf( Prey, N))),
            W,
            S ))
```

Program 1: The best program generated by GP.

prey and the longer they stay there, the higher is the fitness of the strategy. This term favors programs which produce 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 differentially evaluates situations where one or more predators surround the prey.
- Finally, if a simulation ends in a capture position, an additional amount of 4∗
  Number of moves allowed∗Grid width is added to the fitness of the program. This term strongly biases the evolutionary search towards programs that enables predators to maintain their positions when they succeed in capturing a prey.

Distance between agents is measured by the Manhattan distance (sum of x and y offsets) between their locations.

To generate general solutions, i.e. ones that are not dependent on initial predator-prey configuration, the same k training cases are run for each member of the population per generation. The fitness measure is then an average of the training cases. *These training cases can be either the same throughout all generations or randomly generated for each generation.*

4 Experimental results

The GP system, GPengine, used in this research is an extension of that used in [4] and is written in C. A graphical reporting system was created for X–Windows using the Tcl and Tk toolkit [13], with the Blt extension; this system was a modification of that by Martin [11].

As to be expected, the initial randomly generated programs are extremely poor strategies. The GP, however, was successful in evolving increasingly effective strategies over the run as evidenced by the improvement in average and maximum fitness of structures in the population as the run progresses (see Figure 2). Fluctuations in the fitness occur over the entire run because the random initial configurations change between successive generations. Generation 412 had the Best Program which contains 61 nodes and has a fitness of 46660 out of a maximal 48000[1]. This program (structure) is shown in see Program 1.

The moves suggested by this strategy (see Program 1) for various relative positions of a predator with respect to the prey is graphically represented in Figure 3. It is interesting to note how the agents converge on the prey using the policy. More significantly, the solution is stable, in that once the prey is captured, no one makes a move that allows the prey to escape (as opposed to Korf's *max norm* metric solution to the orthogonal game). This figure also shows the corresponding action choices if predators were to follow a deterministic version of Korf's algorithms using *max norm* and *Manhattan distance* metrics respectively. In Korf's version,

[1] This program can be pruned in size without loss of functionality, e.g., the second `IFTE` call has a true condition, and hence always returns the "then" part of the if-then-else structure. As such, this `IFTE` construct can be replaced by just the "then" part without any functional change in the program. We, however, decided to present the actual structure found by the GP.

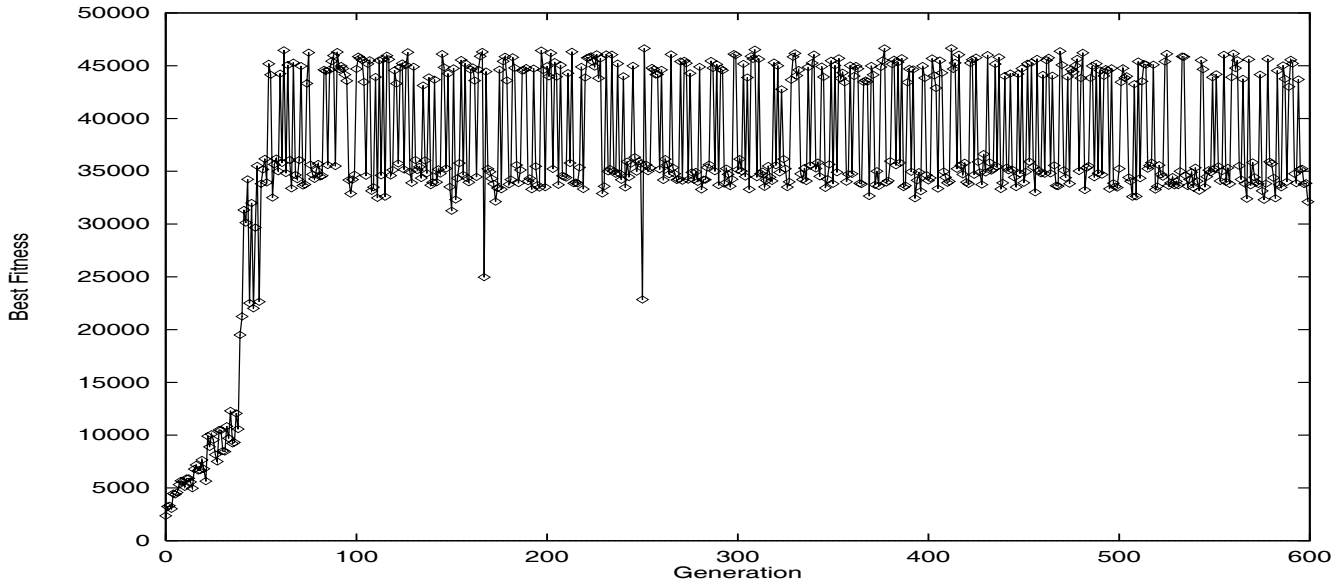

Figure 2: Example fitness curve.

ties are broken randomly, whereas in the figure, ties are broken deterministically by following the order in the word NESW, i.e, in case of a tie between North and East moves, North is chosen, etc. It is hard to tell which of the strategies will fare better on average just by looking at these three figures. The point is that the GP seems to have produced a very plausible strategy using very little domain information. Furthermore, this approach does not rely on any communication between agents and predators are only assumed to be cognizant of the location of the prey (and not that of the other predators).

The program developed by GP, and predator movements using the *max norm* (MN) and *Manhattan distance* (MD) metrics were run for 200 steps on 30 test cases used by Stephens [17], and also on a further set of 1000 randomly generated configurations; average number of captures for each of the algorithms are presented in Table 3. Running the experiments for larger number of steps will raise the capture rate of all the algorithms, but we do not expect their relative performance to change. For each of the randomly generated configurations, experiments were run with two types of prey (one choosing randomly at each time step to move in one of the four directions or to stay still, whereas the other chose a step that will take it maximally away from the nearest predator (MAFNP prey)) and two modes of action choice (in one mode the prey moved first followed by the predator, in the other all agents chose their moves synchronously). Since the agents choose some moves randomly (either because it is a random prey, or because ties are broken randomly), data in the table is averaged over 26 runs[2] for the 30 test cases and 10 runs for the 1000 test cases. The table also contains data from experiments with Korf's original algorithms where the predators can see each other and the prey moves followed by each of the predator's in sequence; the *max norm* and *Manhattan distance* cases are labeled MNO and MDO respectively.

From the table we can observe some general trends which have few exceptions. The following observations can be made from the table:

- Except for the *max norm* metric algorithms, MN and MNO, the prey once captured cannot escape. For the these algorithms we have listed two results, one in which the prey is in a capture position at the end of the run, and another in which the prey was captured anywhere within a run (we call this *shadow capture*). We will use the latter numbers for most of the following discussion, but use the former for rating the different strategies.

- Manhattan distance metric based algorithm beats max norm metric based algorithm (except that MNO beats MDO when random prey moves at the same time as the predators).

- Prey moving at the same time results in more capture than prey moving before predators move (except for the MD case and when the prey is moving

[2]This apparently uncommon number was used for averaging so that we can use the Wilcoxon matched pair signed-rank test [14] to test the significance of the results.

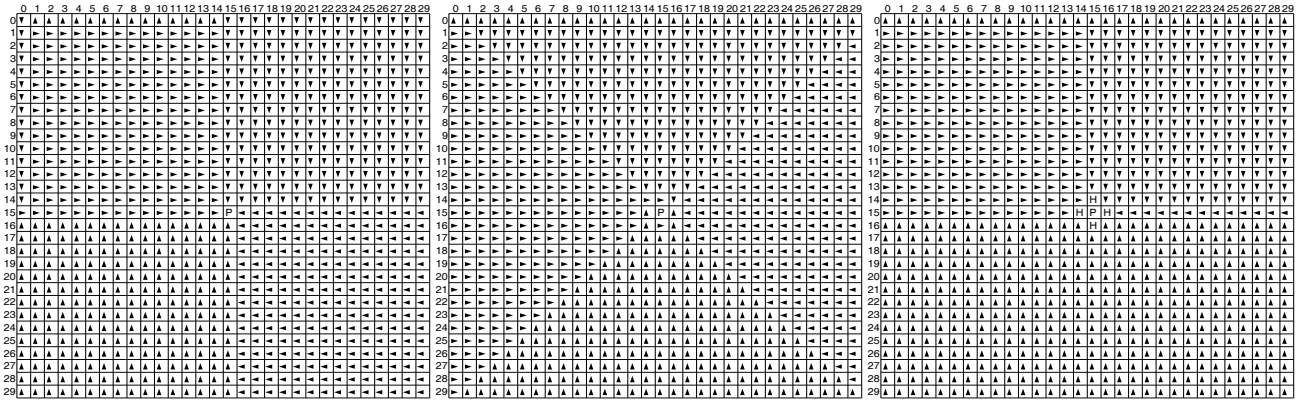

Figure 3: Example pursuit paths found by (a) STGP, (b) MN, and (c) MD.

away from the nearest predator). To explain this trend, let us consider Figure 4(a). If the prey R moves first, it can escape. If all agents move simultaneously, however, both R and 4 may decide to move into the same location and hence will get bounced back to the current locations. This sequence will be repeated until the prey randomly decides to stay still, at which time it will be captured as 4 will move into its only free neighboring location.

- A randomly moving prey is captured more frequently than a prey moving away from the nearest predator (except for MD and MDO when prey moves first). Analyzing the average number of steps taken to capture the prey (data not shown in Table 3), we find that in most cases a random prey is captured more quickly (except for MNO and for GP and MDO when prey moves at the same time as the predators). Korf claimed that a random prey is easier to capture than one which moves away from the nearest predator. We must, however, emphasize that this is true only if the prey decides to stay still with some significant frequency. In Korf's experiments a prey stood still 10% of the time, in ours that frequency is 28% (staying still in 10% of the moves and choosing to stay still randomly as one of 5 choices for the rest 90% of the moves). We performed some additional experiments on the 30 test cases in which the prey randomly chose to move into one of its adjacent open positions, and found that the performance of all the algorithms deteriorated with a noticeable decrease in the case of the prey moving at the same time as the predators.

- Our implementations of Korf's algorithms significantly outperforms the corresponding original formulations by Korf (MN outperforms MNO, and MD outperforms MDO). We view this improvement to have additional significance because MN and MD do not require that predators can see other predators or that they take turns in moving (as required by MNO and MDO). This improvement in performance is due to the fact that we allow agents to try to move into the location contained by another agent. Figure 4 explains how it leads to permanent capture by MN which is difficult to maintain using MNO. It also shows why our assumption of predators being able to push other stationary predators allow predators using the MD algorithm to escape from stalemate situations experienced by predators using the MDO algorithm. In addition, this allows for more efficient pursuit of the prey. In Figure 4(c), agents following MDO strategy will stand their grounds while the prey may move South (thus effectively increasing distance from all predators). When, agents follow MD strategy, however, both 3 and 4 will move South which effectively means that 3, 4, and 2 move South (as 4 will push the stationary 2 South), and only 1 will be further off from the prey after this set of moves.

- When averaged over all scenarios for the 30 or 1000 test cases, the ranking of the algorithms in decreasing order of actual capture rates is as follows: MD, GP, MDO, MN, MNO (the average shadow captures of MNO is greater than the average actual capture rate of MN).

5 Conclusions

We have used Genetic Programming to evolve cooperation strategies for predators to capture a prey moving

	Stephen's 30 test cases				1000 random test cases			
	MAFNP Prey		Random Prey		MAFNP Prey		Random Prey	
	Prey first	Synch.	Prey first	Synch.	Prey first	Synch.	Prey first	Synch.
GP	2.50(1.42)	4.89(1.99)	2.73(1.34)	9.92(2.91)	74.10(9.79)	162.00(10.01)	100.40(13.23)	332.00(18.71)
MN	0.00(0.00)	0.46(0.76)	0.12(0.33)	0.08(0.27)	0.20(0.63)	0.10(0.32)	3.80(1.75)	2.40(1.27)
	0.65(0.89)	0.12(0.33)	10.08(2.17)	6.15(1.83)	31.50(5.19)	4.70(2.36)	352.20(11.25)	196.20(9.65)
MD	18.23(3.02)	7.85(2.36)	22.73(2.75)	20.35(2.59)	610.80(15.21)	276.10(10.06)	759.00(11.94)	681.10(8.95)
MNO	0.00(0.00)	0.00(0.00)	0.04(0.20)	0.00(0.00)	0.10(0.32)	10.00(3.13)	2.40(0.97)	2.80(1.69)
	0.58(0.76)	0.58(0.70)	8.27(2.71)	6.81(2.15)	14.90(3.18)	13.30(3.83)	261.30(13.78)	266.80(11.71)
MDO	2.42(1.21)	2.31(1.29)	0.85(0.83)	1.54(1.36)	86.60(6.48)	88.30(7.01)	43.30(7.96)	42.10(6.19)

Table 3: Average number of captures over different runs (standard deviations are presented in parentheses).

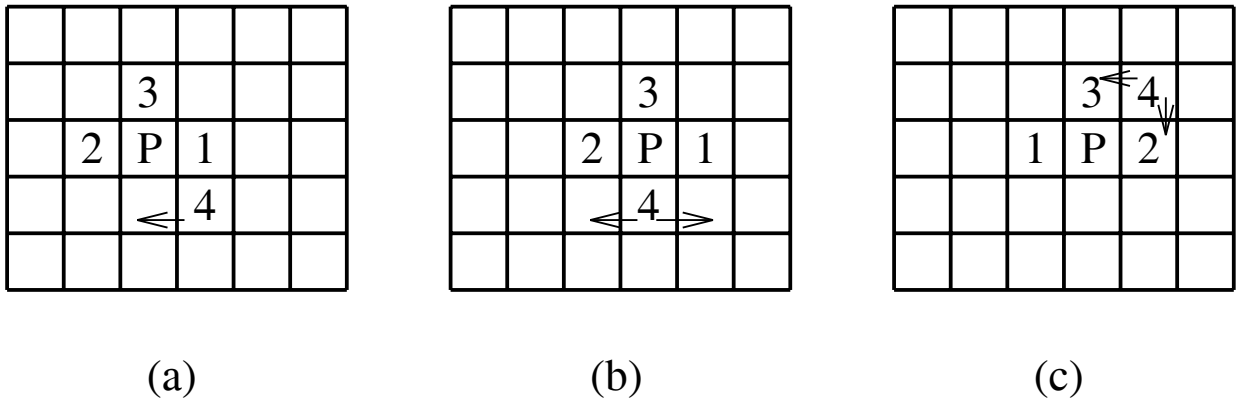

Figure 4: (a) Using MD, agent 4 will move as shown, but using MNO, 4 with equal probability chooses to stay where it is or moves as shown (allowing the prey to escape 50%) of the time. (b) Using MNO or MN, 4 will stay where it is, or move East or West allowing prey to escape. Using MDO or MD, 4 will stay where it is, maintaining a capture position. (c) Using MDO, 4 will retain its current location. Using MD, prey will move South or East; if the latter is chosen, a more conducive situation for capture is produced.

in a grid-world. We have also implemented improved versions of greedy heuristic strategies for capturing the prey. Actually the improvement is obtained by changing assumptions about the environment. Korf's claim [7] that a simple solution exists for the predator prey problem can be challenged by the experimental results presented in Table 3. In particular, we do not observe significantly high capture rates. Also, Korf's original algorithms based on the max norm and Manhattan distance metrics, MNO and MDO, are outperformed by our modified algorithms MN and MD, as well as by the program evolved by GP. Results from both the 30 test cases used in a previous study, and on additional 1000 randomly generated test cases show that the solution evolved by GP is very competitive with handcoded algorithm, and looses out only to the MD algorithm. We are still analyzing the reason for this difference in performance, as the pictorial representations of the strategies (see Figure 3) provides little clue to explain this effect.

These results suggest that the predator-prey domain is not a closed-and-shut case. It still provides interesting challenges to experiment with new multi-agent coordination techniques. The impact of agent capabilities: sensing, noise handling, communication, simultaneous actions, etc. can be gainfully studied by varying different parameters in this domain. Even risking the stigma of being labeled the "blocks-world" problem of DAI, we would like to advocate more use of this easy-to-describe but difficult-to-solve domain as a testbed for comparing and contrasting different coordination mechanisms.

We have shown in this paper that the GP paradigm can be effectively used to generate apparently complex cooperation strategies without being provided with any deep domain knowledge. The evolved strategies fared extremely well compared to some of the best manually constructed strategies. We believe this approach to developing coordination schemes holds further promise for domains in which a human designer has much less understanding of what a good coordination strategy would be.

In place of using a static strategy used by the prey, we plan to perform experiments where the predators and prey are co-evolved. We expect a number of sur-

prises in terms of the types of attack/evasion strategies discovered, and how they trigger each other.

In the current work, we have provided the Manhattan distance measure as a function to the GP program. It would be instructive to see if GP can evolve this necessary functionality from the more elementary function and terminal sets given below:

1. x–coord of Agent A in the Tack B
2. y–coord of Agent A in the Tack B
3. +, -, *, and /.

References

[1] 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.

[2] Lawrence Davis, editor. *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York, NY, 1991.

[3] 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.

[4] Thomas D. Haynes and Roger L. Wainwright. A simulation of adaptive agents in a hostile environment. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, pages 318–323. ACM Press, 1995.

[5] John H. Holland. *Adpatation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[6] Kenneth E. Kinnear, Jr., editor. *Advances in Genetic Programming*. MIT Press, Cambridge, MA, 1994.

[7] 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.

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

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

[10] 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.

[11] Martin C. Martin. graphs.blt. GP FTP Archives, 1994.

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

[13] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, 1994.

[14] L. Sachs. *Applied Statistics: A Handbook of Techniques*. Springer-Verlag, 1982.

[15] Munindar P. Singh. 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*, October 1990.

[16] 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.

[17] 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.