# Evolving Multiagent Coordination Strategies with Genetic Programming

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**

The design and development of behavioral strategies to coordinate the actions of multiple agents is a central issue in multiagent systems research. We propose a novel approach of evolving, rather than handcrafting, behavioral strategies. The evolution scheme used is a variant of the Genetic Programming (GP) paradigm. As a proof of principle, we evolve behavioral strategies in the predator–prey domain that has been studied widely in the Distributed Artificial Intelligence community. We use the GP to evolve behavioral strategies for individual agents, as prior literature claims that communication between predators is not necessary for successfully capturing the prey. The evolved strategy, when used by each predator, performs better than all but one of the handcrafted strategies mentioned in literature. We analyze the shortcomings of each of these strategies. The next set of experiments involve co–evolving predators and prey. To our surprise, a simple prey strategy evolves that consistently evades all of the predator strategies. We analyze the implications of the relative successes of evolution in the two sets of experiments and comment on the nature of domains for which GP based evolution is a viable mechanism for generating coordination strategies. We conclude with our design for concurrent evolution of multiple agent strategies in domains where agents need to communicate with each other to successfully solve a common problem.

## Keywords

Multiagent Coordination, Behavioral Strategies, Evolution of Behavior, Genetic Programming

## 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) [4]. 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 [11], 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.

We believe that evolution can provide a workable alternative in generating coordination strategies in domains where the handcrafting of such strategies is either prohibitively time consuming or difficult.

The approach proposed in [11] for developing coordination strategies for multi–agent problems is completely domain independent, and uses the strongly typed genetic programming (STGP) paradigm [23], which is an extension of genetic programming (GP) [18]. 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 both single and multiagent domains domains to evaluate the evolution of behavioral strategies by STGP [11, 12]. Of particular relevance to this special issue is the predator–prey pursuit game [3]. We have used this domain to test our hypothesis that useful coordination strategies can be evolved using the STGP paradigm for non–trivial problems. This domain involves multiple predator agents trying to capture a prey agent in a grid world by surrounding it. The predator–prey problem has been widely used to test new coordination schemes [7, 17, 20, 29, 30]. 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 find that STGP evolved coordination strategies perform competitively with the best available manually generated strategies in this domain. Though the rest of this paper concentrates on the predator–prey domain, we will emphasize at each point the general multiagent problem solving issues being addressed by this research.

The rest of this paper is organized as follows: Section 2 provides a history of the predator–prey domain. Section 3 introduces both GP and STGP. It also details our approach for representing the predator–prey domain. Section 4 presents experimental results in the evolution of coordination strategies by STGP. The best strategy evolved by STGP is compared to several handcrafted strategies reported in previous literature. Section 5 presents experimental results from our attempt to competitively co–evolve both predator and prey strategies. Most importantly it analyzes why greedy predator strategies fail against the prey strategies produced by co–evolution. Section 6 presents our analysis of the capabilities needed to capture the prey which moves in a straight line. Section 7 presents our conclusions about the utility of GP for evolving behavioral strategies.

## 2 The Pursuit Problem

The original version of the predator–prey pursuit problem was introduced by Benda, *et al.* [3] 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.* [7] 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 [27] on using group intentions for agent coordination, lacks any experimental results that allow comparison with other work on this problem.

Stephens and Merx [29, 30] 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 [17] 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. [21] 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. They found that the $4 \times 1$ 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.

# 3 Evolving Coordination Strategies

In the following subsections we briefly introduce the genetic programming paradigm, along with its strongly typed variant, and explain how we have used it to evolve coordination strategies.

## 3.1 Genetic Programming

Holland's work on adaptive systems [14] 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 [5].

Koza's work on Genetic Programming [18] 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 [15, 18, 19].

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 [23] 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.

## 3.2 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 evolved behavioral strategies to be used by the predator agents. Following Korf's lead, each strategy is tested by using it to control the actions of each predator.

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. Our goal 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

## 3.3 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

| 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

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 with the STGP scheme, 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 Evolution of Individual Greedy Strategies

Korf's [17] basic claim is that predators need not jointly decide on a strategy, but can choose locally optimal moves and still be able to capture the prey consistently. We used this philosophy as the basis for our research in this domain. Like Korf, our predators choose their moves individually, i.e. without reasoning about the

moves of other predators or without mutually deciding on a set of moves. Our goal is to find out if effective individual behavior can be evolved in multiagent scenarios which require multiple agents to contribute for the success of the group.

For our experiments, a STGP algorithm is put to the task of evolving a program that is used by a predator to choose its moves. The same program is used by all four predators in a simulation. Each program in the population, therefore, represents a strategy for implicit cooperation to capture the prey.

The initial configuration consists of the prey in the center of a 30 by 30 grid and the predators placed in random non–overlapping positions. All agents choose their action simultaneously. The environment is accordingly updated and the agents choose their next action based on the updated environment state. If two agents try to move into the same location simultaneously, they are "bumped back" to their prior positions. One predator, however, can push another predator (but not the prey) if the latter decided not to move. The prey does not move 10% of the time; effectively making the predators travel faster than the prey. The grid is toroidal in nature, and the orthogonal form of the game is used, i.e. agents can not move horizontally. A predator can see the prey, and the prey can see all the predators. Furthermore, the predators do not possess any explicit communication skills; two predators cannot communicate to resolve conflicts or negotiate a capture strategy.

We decided to utilize STGP rather than GP because STGP reduces the solution space to be searched [11, 23]. The GP system, GPengine, used in this research is an extension of that used in [13] and is written in C. Furthermore, it allows for strong typing, as described by [23]. A graphical reporting system was created for X–Windows using the Tcl and Tk toolkit [24], with the Blt extension; this system is a modification of that by Martin [22].

## 4.1 Deterministic Predator Algorithms

Korf [17] used two greedy heuristics in his work: *Manhattan distance* and *max norm*. The *Manhattan distance* algorithm determines the best move to make based on the sum of the differences of the x and y coordinates of a predator and the prey. The *max norm* algorithm determines the best move to make based on the diagonal distance, i.e. maximum of the differences of the x and y coordinates, between a predator and the prey.

We have modified these algorithms to fit into our definition of the domain. Our predator algorithms, *Manhattan distance* (MD) and *max norm* (MN) follow the rules outlined in Section 2. They also move at the same time, in contrast to the predators in the original algorithms, *Manhattan distance original* (MDO) and *max norm original* (MNO), which take turns in moving, and thus have no conflict for cells.

## 4.2 Deterministic Prey Algorithms

The prey algorithms presented in the literature are a randomly moving prey (Random) and a move away from the nearest predator prey (MAFNP). In Korf's [17] formulation of the randomly moving prey, the prey's possible moves are limited to those surrounding cells which are unoccupied by the predators. Since we chose to let all agents, prey and predators, move simultaneously, the randomly moving prey is allowed to consider all directions. In Korf's setup, if the prey was surrounded on three sides, it would move to the free cell. In our setup, there is a 28% probability that the prey will choose the free cell. This is because it will stay still 10% of the time, and for the other 90% of the time all directions are equally likely.

The MAFNP prey determines the set of predators closest to it. It then randomly picks one, say $P_i$, to move away from. The set of moves which maximizes the distance from $P_i$ is constructed, and one is chosen randomly. The prey will then move in the corresponding direction.

## 4.3 Evolved Predator Strategy

A typical result for a STGP run with a randomly moving prey is shown in Figure 1. This fitness curve indicates that good building blocks or subprograms are being identified. As expected, the initial randomly generated programs were extremely poor strategies. The STGP, however, was successful in evolving effective strategies over the run as evidenced by the improvement in maximum fitness of structures in successive populations (see Figure 1). Fluctuations in the fitness occur over the entire run because the random initial

configurations change between successive generations. Generation 412 had the best program (see Program 1) which contains 61 nodes and has a fitness of 46660 out of a maximum possible fitness of 48000.

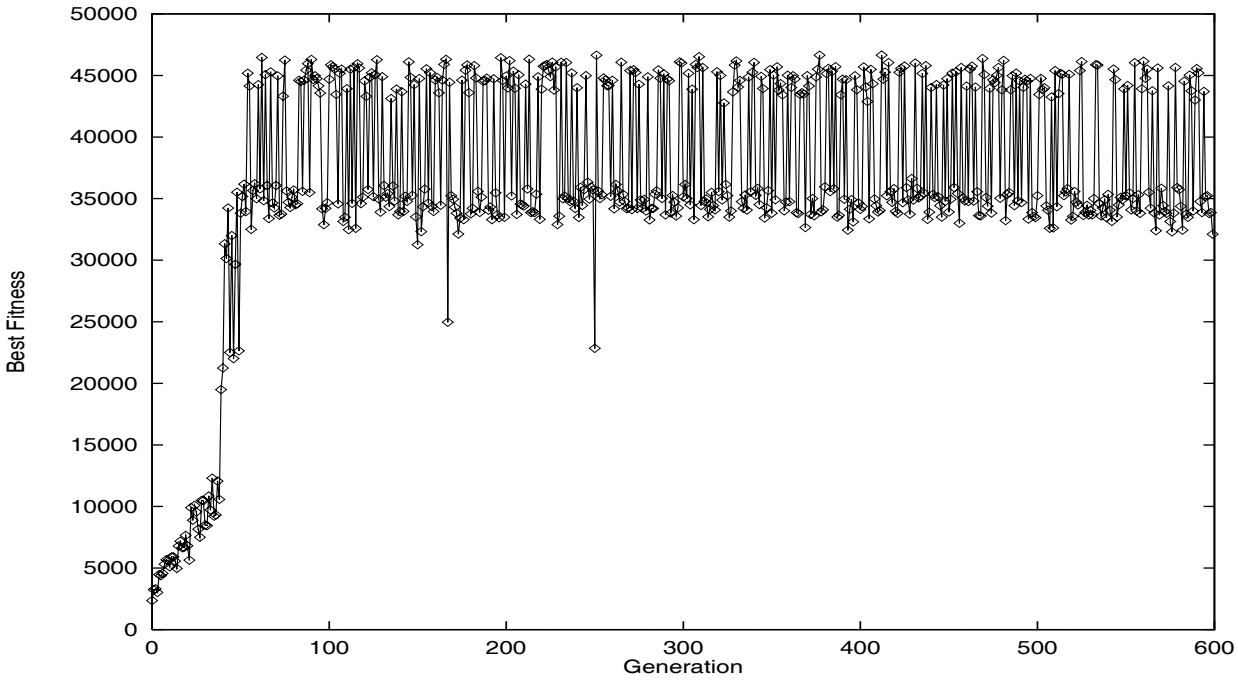



Figure 1: Best Fitness curve for STGP predators against a Random prey.

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 2(a). Figure 2(b) and Figure 2(c) show the corresponding moves taken by MNO and MDO respectively. The arrows denote the direction a predator would like to move, the "P" denotes the prey, and the "H" denotes *Here*, which means the predator wants to stay still.

It is interesting to note how the agents converge on the prey using the STGP policy: a predator advances to an orthogonal axis and then close in on the prey. It is important to note that the STGP solution is stable, in that once the prey is captured, no predator makes a move that allows the prey to escape. The STGP produces a very plausible strategy using very little domain information. Furthermore, this approach does not rely on any communication between agents and hence imposes no cognitive burden. Predators are only assumed to be cognizant of the location of the prey and need not even sense the location of the other predators.

The STGP algorithm is deterministic in its choice of direction, while the MNO and MDO algorithms can choose non–deterministically from a set of equally good alternatives. (see subsection 4.1). Therefore the directions shown, in Figure 2(b) and Figure 2(c), are the first member of a set of equally good alternatives, with the ordering being *North*, *East*, *West*, *South*, and *Here*. It is also interesting to note that the MNO algorithm cannot produce stable captures. The STGP algorithm leads the predators to "push" against the prey, and the MDO algorithm has the predators stay in the capture positions once they occupy these positions. The MNO algorithm forces the predators to mill around the prey, which allows the prey to escape.

## 4.4 Experimental Results

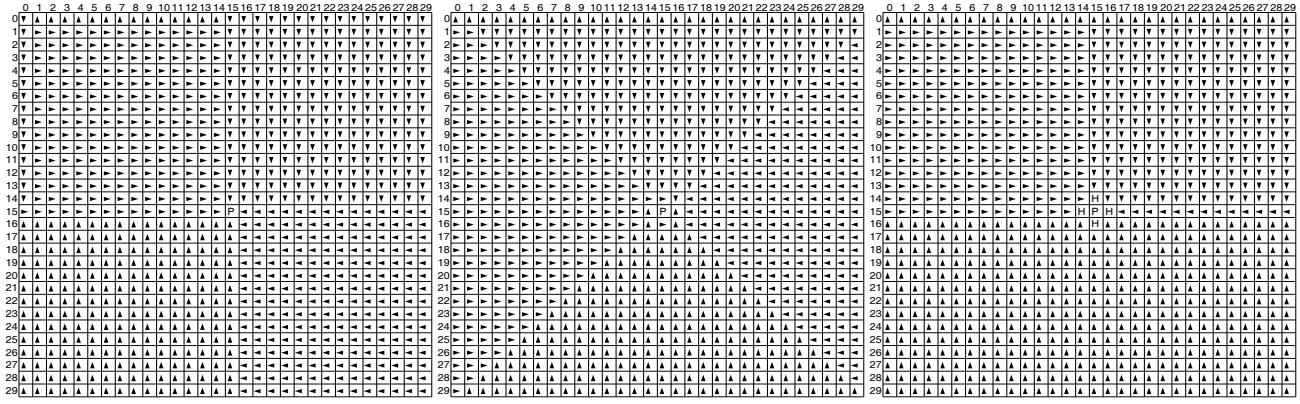
An important question is how does the best evolved predator behavioral (STGP) strategy compare against the handcrafted predator algorithms? The answer is quite favorably.

To compare the algorithms we utilized 30 test cases from Stephens [29], averaged over 26 different initial random seeds.[1] In our previous work [10, 11] we had allowed the simulation to run for 200 time steps. We

[1] This apparently uncommon number was used for averaging so that we can use a formula for the Wilcoxon matched pair signed-rank test [26], which is used to test the significance of the results.

```
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 STGP.



(a) (b) (c)

Figure 2: Example pursuit paths found by (a) STGP, (b) *max norm*, and (c) *Manhattan distance*.

found that MNO and MDO did not have the capture rates reported by Korf, who did not report the maximum number of time steps used in his experiments. In an effort to remove any unfair advantage our simulations would have, we increased the number of time steps to 2000. We feel that this limit is both practical and realistic. This increase in maximum time steps leads to a corresponding increase in the capture rate from our previous work.

While evolving the best STGP strategy, we let the prey move at the same time as the predators. Korf had the prey move first followed by the predators taking turns. In another effort to compare our algorithms to those originally presented by Korf we ran a set of tests where the prey moves before the predators.

Table 3 presents the results from allowing the prey to move with the Random algorithm. The most striking observation is that both MNO and MN have very poor capture rates. The answer for this lies in Korf's definition of capture versus our definition. He stops a simulation if the predators surround the prey. We, however, require the captures to be stable. In order to fairly represent *shadow captures* (captures at any point in the simulation) by MN and MNO, we include an additional line in the tables for these strategies. Note that STGP, MDO, and MD are all strategies in which once the prey is captured, it never escapes.

Table 4 presents the results from allowing the prey to move with the MAFNP algorithm. Combined with the results for the Random prey, Table 3, we observe that in general, the predator algorithms can be ranked as: MD, STGP, MDO, MNO, and MN. A surprising result of these tests is shown in Figure 3 and Figure 4, which represent shadow captures against the Random and MAFNP prey algorithms respectively. Namely,

the MD algorithm is consistently better than both MN and MNO. This is contrary to what Korf has claimed.

We believe that the better capture rate is a direct result of the inability of the *max norm* metric algorithms to stay in a capture position, as shown in Figure 2(b). There is a probability of 33% that the *max norm* metric algorithms will stay in a capture position. We further elaborate on this problem of the *max norm* metric in Section 5.2. In Figure 2(c), we find that the *Manhattan distance* metric algorithms will always stay in the capture positions. The *max norm* metric algorithms have higher capture rates for a Random prey than for a MAFNP prey. This stems from the Random prey having spatial locality, which means that when the predator non–deterministically moves away from a capture position, it is likely to be able to move back into a capture position. With the MAFNP prey, leaving a capture position can result in a chase to re–occupy a capture position.

From Table 4 and Table 3, we see that the Random prey is easier to capture than the MAFNP prey. Either the capture rate improves, as exemplified by MN and MNO, or the steps to capture decreases, as shown by MD. The Random prey being easy to capture follows Korf's claims.

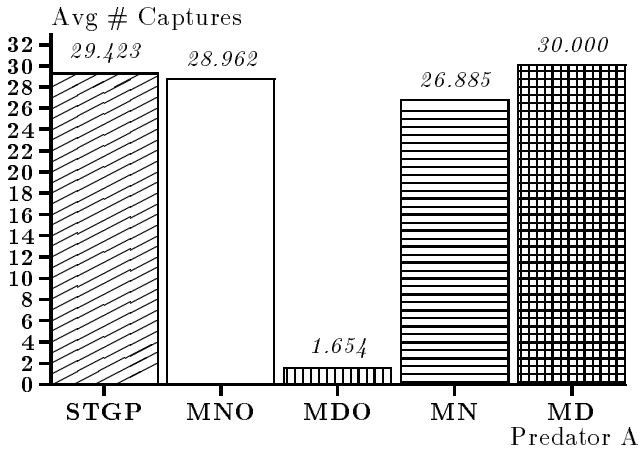



Figure 3: Average number of shadow captures for Random prey, moving with the predators.

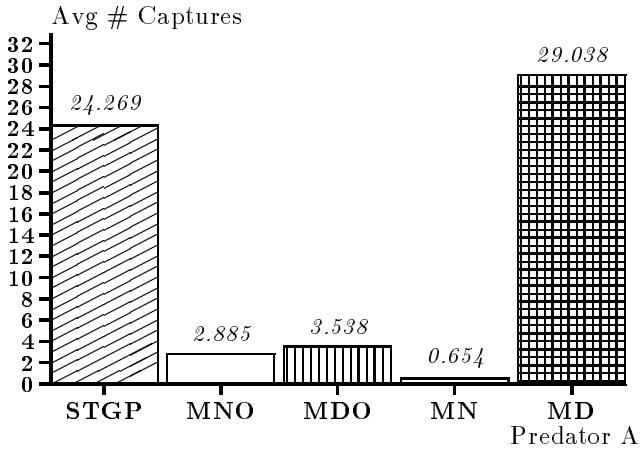



Figure 4: Average number of shadow captures for MAFNP prey, moving with the predators.

## 4.5 Analysis of Evolution

The basic question that arises from these tests is why does the MD strategy capture the prey more than the STGP strategy? In Figure 5, we see that **predator 3** and **predator 2** are both contending for the same cell, i.e. they are in a deadlock situation. If the predators are employing the STGP strategy, then the deadlock is

| | Prey First | | | Prey Sync | | |
|---|---|---|---|---|---|---|
| | Captures | Steps | Blocks | Captures | Steps | Blocks |
| STGP | 15.808(2.912) | 844.268(152.588) | 5.269(1.909) | 29.423(0.809) | 449.190(91.864) | 0.308(0.549) |
| MNO | 0.077(0.272)<br>29.038(0.958) | 1999.000(1958.612)<br>529.781(65.943) | 9.500(2.596) | 0.154(0.368)<br>28.962(0.916) | 1998.500(1874.759)<br>542.256(63.061) | 9.269(2.342) |
| MDO | 1.731(1.116) | 296.445(315.689) | 21.115(2.197) | 1.654(1.198) | 227.116(318.616) | 21.423(2.656) |
| MN | 0.077(0.272)<br>29.654(0.629) | 1999.000(1958.612)<br>405.008(71.496) | 9.115(3.290) | 0.115(0.326)<br>26.885(1.558) | 1999.000(1917.373)<br>662.238(91.512) | 6.038(2.341) |
| MD | 30.000(0.000) | 151.233(25.756) | 0.000(0.000) | 30.000(0.000) | 180.242(32.011) | 0.000(0.000) |

Table 3: Average number of captures for Random prey (standard deviations are presented in parentheses).

| | Prey First | | | Prey Sync | | |
|---|---|---|---|---|---|---|
| | Captures | Steps | Blocks | Captures | Steps | Blocks |
| STGP | 13.692(2.429) | 826.742(176.856) | 0.962(1.428) | 24.269(1.909) | 730.799(107.086) | 0.038(0.196) |
| MNO | 0.000(0.000)<br>3.077(1.896) | 0.000(0.000)<br>939.075(456.787) | 0.269(0.452) | 0.000(0.000)<br>2.885(1.705) | 0.000(0.000)<br>928.227(490.409) | 0.269(0.533) |
| MDO | 3.885(1.633) | 213.347(207.632) | 16.577(2.996) | 3.538(1.581) | 216.859(150.604) | 16.769(2.847) |
| MN | 0.000(0.000)<br>7.038(2.068) | 0.000(0.000)<br>947.891(268.332) | 0.885(0.909) | 0.000(0.000)<br>0.654(0.797) | 0.000(0.000)<br>837.118(738.445) | 0.077(0.272) |
| MD | 30.000(0.000) | 228.301(43.916) | 0.000(0.000) | 29.038(1.076) | 557.981(64.638) | 0.000(0.000) |

Table 4: Average number of captures for MAFNP prey (standard deviations are presented in parentheses).

never resolved, because each of the predators continues to bounce off of each other. However, if the predators are employing the MD strategy, then eventually **predator 3** will non–deterministically decide to move *South*. This illustrates the basic difference between these two strategies. On closer inspection, the STGP evolved strategy reveals itself as a special case of the MD strategy in which one of the move options available to the MD strategy is deterministically preferred over the other options.

In summary, these set of experiments demonstrate that evolution is a viable mechanism for generating individual behavior strategies that lead to effective group performance. This is particularly illuminating because both little domain knowledge is provided and we do not allow for explicit communication between agents in the group. This bodes well for the possibility of evolving coordination strategies for other multiagent domains.

# 5 Competitive Co–evolution

In our initial experiments on evolving coordination strategies for predator agents in the predator–prey domain, the STGP paradigm was able to evolve a program which had a better strategy than all but one of four manually derived greedy algorithms. In the belief that the static program of the prey was limiting the search for a better program, we decided to explore coevolving cooperation strategies in a predator population and avoidance strategies in a prey population. The basic premise of co–evolution is that if one population devises a good ploy, then the other population will construct a counter to that ploy. We expect that the populations will see–saw between being better on the average. This has been shown in Reynold's work on co–evolution in the game of tag [25]. In his work, the two opposing agents, from the same population, take turns being

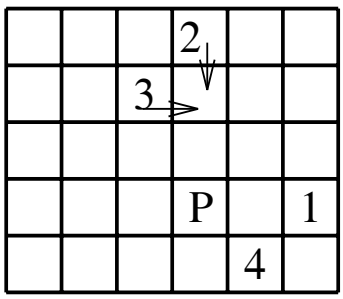



Figure 5: Deadlock scenario for STGP but not MD.

the predator and the prey. Whereas in our work, there are separate populations and the predator population has to manage cooperation between multiple agents.

## 5.1 Evaluation of Prey Strategies

In each simulation there is a maximum number of fitness points that the predator program can be awarded. As they strategy employed by the prey improves, the score attained by the predator will decrease. Therefore the prey can be judged by the reverse criteria presented in Section 3.3, i.e. it is trying to maximize the distance between itself and the predators.

## 5.2 Experimental Results

During the course of this experiment, we did not see the results that we were expecting. The predators seemed incapable of even moving towards the prey. This corresponded to a high fitness level for the prey. There were two main differences from our previous efforts to generate coordination strategies: the predators could see each other, and the prey was learning to escape the predators.

We had enhanced the language of the predators, to allow them to see each other, for two reasons: the predators and prey would share the same languages, and the predators would be capable of more explicit cooperation. We investigated the predators being able to see each other by evolving strategies in a given prey environment. We found the evolved strategies still ignored other predator locations.

In order to test the hypothesis, that the prey was learning to escape the predators, we ran experiments where the prey was pitted against our version of *Manhattan distance* (MD) algorithm [10]. The prey was very successful in evading the predators. This was particularly surprising because the algorithm developed by the prey was simple: pick a random direction and move in a straight line in that direction. This prey algorithm, which we will refer to as the Linear prey, produced less capture than that found with all the previously examined prey algorithms. Note that we can design more sophisticated prey algorithms that will do even better, but *a priori* such a simple scheme would not have been given much of a chance against the given predator algorithms. Also, the Linear prey does not need to sense the predator locations, which makes it easier to use than more sophisticated strategies.

The results for the Linear prey are shown in Table 5. To be fair to the predator algorithms, we executed each training case with the prey moving in each of the four directions. Thus 3120 test cases were considered instead of the 780 test cases presented in the other result tables. All of the predator algorithms performed poorly at capturing the Linear prey.

A question which arose at this point was how often does a Linear prey get blocked? Our thinking was that the reason the Linear prey was so successful in evading capture was that no predator was able to block the prey's movement, which in turn would allow the other predators to enact a capture. Until this point in our experiments, we had not been concerned with *blocks*. A block is defined as a test case in which the prey was not captured and a predator occupied the cell to which the prey wanted to move on its last turn. Interestingly the algorithms based on the *Manhattan distance* metric: STGP, MDO, and MD, were more successful at blocking the Linear prey from moving than were the algorithms based on the *max norm* metric. If we examine the basic premise of the *max norm* metric algorithms, then we realize that while a predator may block the prey at some point in the simulation, it is very likely to move away, allowing the prey to escape. The sequence of movements in Figure 8 depict such a scenario. The *Manhattan distance* metric algorithms allow the predators to stay in the blocking positions.

The *Manhattan distance* metric algorithms are able to block the Linear prey more than half of the time, so why do they not capture the Linear prey once it is blocked? To answer this question, we experimented with a prey that does not move, which we will call the Still prey. In Table 6 we see that the *max norm* metric algorithms are successful in shadow capturing the Still prey, but the *Manhattan distance* metric algorithms do not fare as well. This data is representative of the scenario of a Linear prey being blocked.

The problem that develops is that one predator agent is unable to move around another predator to get to the prey, as shown in Figure 6(c). Since the *Manhattan distance* metric algorithms are greedy and the predators choose their moves without consulting each other, these deadlock situations will not be resolved. Note that Korf was motivated to use the *max norm* metric to solve this particular problem. Using the *max norm* metric in the simulation depicted in Figure 6(c), **predator 2** will eventually move *North* or *South*,

allowing **predator 3** to move in on the prey. But we have seen before that such behavior also prevents predator agents using the *max norm* metric from successfully capturing the prey in many more situations.

A broader issue that the Still prey raises is why is it harder to capture than a Random or MAFNP prey? (See Tables 3, 4, 6.) Based on the above analysis of deadlock conditions, we believe that the act of the prey moving helps the predators avoid deadlock. The shifting prey changes the directions that the predators wish to take, making a contested cell no longer of interest to some of the agents.

The Linear prey algorithm points to another prey algorithm: maximize distance from all predators (1Ply). The 1Ply algorithm exhaustively computes the sum of the *Manhattan distances*, to the predators, for the possible moves that the prey may chose. The move corresponding to the maximal distance is selected, with all ties being broken randomly. While we do not present the results here, we are aware that once we evolve an algorithm that is successful against the Linear prey, we must then turn our attention to the 1Ply prey. The Linear prey has regularity in its movement, while the 1Ply prey does not.

The initial results from the prey learning experiment also prompted us to conduct an experiment in which predators were trained against a prey which moves in a straight line. The best evolved strategy is referred to as Linear STGP. In [8] we found it to be evident that the Linear STGP strategy is not very general, and only has significant performance when pitted against a Linear prey.

| | Prey First | | | Prey Sync | | |
|---|---|---|---|---|---|---|
| | Captures | Steps | Blocks | Captures | Steps | Blocks |
| STGP | 5.923(1.055) | 23.779(2.061) | 68.000(2.713) | 3.423(1.065) | 23.393(4.652) | 69.731(3.040) |
| MNO | 0.000(0.000)<br>0.538(0.859) | 0.000(0.000)<br>830.786(708.778) | 0.038(0.196) | 0.000(0.000)<br>0.423(0.643) | 0.000(0.000)<br>900.636(752.111) | 0.000(0.000) |
| MDO | 5.577(1.880) | 23.848(2.963) | 67.077(3.285) | 5.192(2.281) | 23.674(3.778) | 66.077(4.009) |
| MN | 0.000(0.000)<br>3.385(1.627) | 0.000(0.000)<br>1125.398(226.410) | 0.192(0.402) | 0.000(0.000)<br>0.577(0.643) | 0.000(0.000)<br>829.333(700.139) | 0.038(0.196) |
| MD | 7.231(2.405) | 36.324(14.832) | 87.115(4.385) | 6.500(2.672) | 39.396(12.153) | 89.654(3.867) |

Table 5: Average number of captures for Linear prey (standard deviations are presented in parentheses).

| | Prey First | | | Prey Sync | | |
|---|---|---|---|---|---|---|
| | Captures | Steps | Blocks | Captures | Steps | Blocks |
| STGP | 3.000(0.000) | 17.333(0.000) | 27.000(0.000) | 3.000(0.000) | 17.333(0.000) | 27.000(0.000) |
| MNO | 0.346(0.485)<br>30.000(0.000) | 1999.000(1648.418)<br>109.194(14.482) | 29.192(0.981) | 0.423(0.578)<br>30.000(0.000) | 1998.818(1599.055)<br>109.680(12.156) | 29.231(0.710) |
| MDO | 2.885(1.336) | 19.480(1.168) | 27.115(1.336) | 2.923(1.164) | 19.658(1.336) | 27.077(1.164) |
| MN | 0.000(0.000)<br>19.846(2.572) | 0.000(0.000)<br>127.494(21.722) | 24.577(1.901) | 0.000(0.000)<br>20.346(3.237) | 0.000(0.000)<br>119.240(19.297) | 25.000(2.059) |
| MD | 4.269(1.041) | 25.577(5.285) | 25.731(1.041) | 3.808(1.132) | 28.455(9.073) | 26.192(1.132) |

Table 6: Average number of captures for Still prey (standard deviations are presented in parentheses).

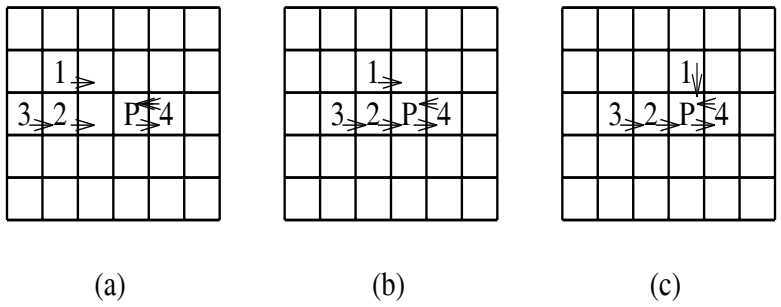



Figure 6: A possible scenario in which a *Manhattan distance* metric based predator tries to block the **prey P**. (a) **predator 4** manages to block **P**. **predators 1**, **2**, and **3** move in for the capture. (b) **predator 2** has moved into a capture position. (c) **predator 1** has moved into a capture position. **predator 2** will not yield to **predator 3**. They are in deadlock, and the **prey P** will never be captured.

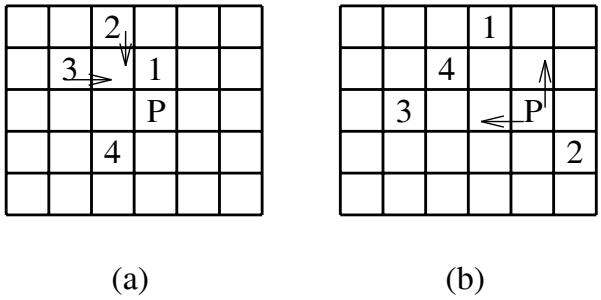



Figure 7: Scenarios showing weaknesses of some prey strategies. (a) Deadlock situation for the predators. (b) MAFNP can lead to capture.

### 5.3 Analysis of Competitive Co–evolution

Some compelling questions arose from these experiments in co–evolution:

1. Why is the Linear prey so effective compared to the other deterministic prey algorithms?
2. Is the encoding of domain responsible for our failure to capture the prey?

The Linear prey is so effective because it avoids locality of movement. The greedy strategies [10, 17] fare well against the preys staying in a small neighborhood. Locality allows the predators both to surround the prey and to escape any deadlock situations, such as that in Figure 7(a). The deadlock is that **predator 2** and **predator 3** are both vying for the same cell. In this situation, **predator 2** can get out of the deadlock with **predator 3** when the prey moves such that the strategy employed by the predators no longer necessitate a move to the cell *South* of **predator 2**. After the deadlock is resolved, the prey will "tend" to be in the neighborhood of where it currently is located. Since the Linear prey avoids locality, the predators tend to "fall" behind the prey, and keeps pursuing the latter forever in the toroidal world.

The MAFNP prey can be captured due to a situation as depicted in Figure 7(b). While moving away from the nearest predator, it allows itself to become surrounded by other, more distant, predators. Since the Linear prey stays ahead of the predators, and the predators greedily move towards the prey, unless a predator is able to block the prey within a number of moves, equal to the grid size, the prey cannot be blocked by a predator.

Our choice of function and terminal sets may be responsible for *our* failure to capture the prey. It is *not* responsible for the failure of the algorithms based on Korf's research: MN, MD, MNO, and MDO. The failure is a result of the agents being greedy.

An important observation from this set of experiments is that evolution can provide surprises that are not easily anticipated when handcrafting solutions to multiagent problems. Co–evolution can provide challenging scenarios that are either overlooked or underestimated at first glance.

## 6 What is needed to Capture?

It is evident that the greedy nature of the predator algorithms precludes the Linear, 1Ply, and Still prey algorithms from being captured. What types of either greedy or non–greedy algorithms will be successful at capturing the prey algorithms mentioned above?

We have tried a different conflict resolution than the bumping discussed in Section 4. Specifically, one of the agents vying for a cell non–deterministically is allowed to enter that cell. The resulting predator agents had a lower capture rate.

We also tried to evolve a predator strategy with an additional function: *Tack* **RanOr**( *Tack* A, *Tack* B). **RanOr** non–deterministically selects either the subtree represented by A or B, and returns the associated *Tack* value. The intent of this function was to create an agent as discussed at the in Section 5.2, i.e. one that combined *Manhattan distance* and *max norm* metrics. The best evolved strategy was poor.

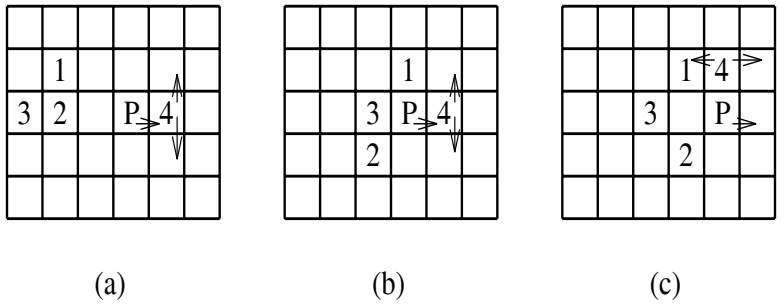



Figure 8: A possible sequence of movements in which a *max norm* metric based predator tries to block the **prey P**. (a) **predator 4** manages to block **P**. Note that **4** is just as likely to stay still as move *North* or *South*. (b) **predators 1** and **3** have moved into a capture position, and **predator 2** is about to do so. Note that **4** is just as likely to stay still as move *North* or *South*. (c) **predator 4** opts to move to the *North*, allowing the **prey P** to escape. Note that **4** is just as likely to stay still as move *East* or *West*.

We have also evolved strategies with the *max norm* metric instead of the *Manhattan distance* metric and strategies that could select either metric. Both of these experiments produced strategies that were no better than the algorithms we have discussed earlier in this paper.

In general, we believe that the predator agents must employ either some explicit communication or further domain knowledge to capture the Linear, 1Ply, and Still prey algorithms. A contract net [28] based predator algorithm, where agents get assigned different sides of the prey to occupy, should fare well against these prey algorithms. Likewise, predator algorithms that take into consideration the toroidal nature of the grid world, and can wait for the prey to wrap around and move into a trap, should also perform well against these prey algorithms.

We are continuing experimentation in evolving strategies to capture these prey algorithms. We have identified two possible domain refinements to help the evolved predators capture the prey: evolving a team instead of an individual and allowing the predators to broadcast their intended capture position.

The developed strategies in this research had implicit communication in that the same program was used to control the four predator agents. We are examining the rise of cooperation strategies without implicit communication [9]. This is achieved by having each predator agent being controlled by its own program. Such a system solves a cooperative co–evolution problem as opposed to a competitive co–evolution problem as described in [2, 8, 25]. We believe that cooperative co–evolution provides opportunities to produce solutions to problems that can not be solved with implicit communication.

# 7 Conclusions

Strongly typed genetic programming is able to generate effective individual behavioral strategies for predator agents trying to capture a prey agent. The evolved strategy is greedy in nature, and is comparable in performance to manually derived greedy strategies. In particular, the evolved strategy captures the prey more often than Korf's original algorithms. Like the *max norm* and *Manhattan distance algorithms*, the STGP algorithm is effective in capturing both a randomly moving prey and a prey which moves away from the nearest predator. Likewise, all of the algorithms fare poorly against a prey that moves in a straight line, a prey that maximizes its distance from the current positions of the predators, and most surprisingly a prey that does not move at all!

In trying to further the utility of the evolved cooperation strategies we ran some experiments in competitive co–evolution. We encountered two obstacles, which have far–reaching consequences: the competitive co–evolution work only partially in the domain, and a simple algorithm for controlling the prey agent was able to successfully evade even the best human derived algorithm for the predator agents. These two obstacles are actually related in that the first is brought about as a result of the second. We believe the reason that competitive co–evolution did not fully succeed in our domain is due to the fact that one population, the prey, quickly found one of the two algorithms capable of confounding the language we had selected for our agents.

The coordination strategies for the predator agents need one of two abilities in order to have a chance of capturing the prey agents: either a means to detect when the local greedy algorithm should yield to the global coordination algorithm or a history mechanism which would enable the predators to not only predict where the prey will move to in the next time step, but also where it will be in $k$ time steps.

The basic problem in this work is the lack of any explicit communication between the agents to resolve conflicts. The agents rely on a simplistic algorithm to resolve conflicts, which combined with greedy nature of the algorithms controlling the predators leads to deadlock situations.

Human agents easily resolve these types of conflicts, so what is needed to allow adaptive agents to model this behavior? If we place four humans in a predator-prey domain, we will see cooperative behavior in the form of allowing one agent to occupy a cell and in agents going around either other agents or the prey in order to effect a capture.

Greedy behavioral strategies are preferable in multiagent domains in which they readily solve the problem at hand. For example, the greedy strategies are sufficient in the predator–prey domain when the problem is to capture either a Random or MAFNP prey. The reason the greedy strategies are preferable is because they impose less cognitive burden on the part of the researcher and/or they can require less skill.

However, this research has shown that even in a simple domain, complex problems can occur in which the greedy strategies are not sufficient in finding a solution. An example in the predator–prey domain is the Linear prey. These more complex problems require either explicit communication or a group effort, i.e. a non–greedy approach.

What is the role of evolution in such complex situations? Or, how does this research carry over into domains other than the predator–prey? Genetic programming is able to quickly search the solution space of complex vocabularies. This search is implicitly parallel, and is speeded up by the ability of the GP paradigm to recombine solutions into increasingly better solutions. A necessary criteria is that the distribution from which the training samples are chosen should be representative of the distribution from which the test samples are drawn. A drawback to the GP paradigm is that without better identification and encapsulation of reusable components, strategies which are complex and encapsulate a large amount of knowledge are unlikely to be evolved. The GP community is actively researching this issue [1, 16, 19].

## Acknowledgments

We must thank Larry Stephens for providing us with the thirty test cases he and Matthias Merx used in their work [29, 30]. It provided us with an opportunity to compare our work with that of previous researchers.

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

## References

[1] Peter J. Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pages 75–97. MIT Press, Cambridge, MA, 1994.

[2] 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.

[3] 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.

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

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

[6] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, March 1989.

[7] 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.

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

[9] Thomas Haynes, Sandip Sen, Dale Schoenefeld, and Roger Wainwright. Evolving a team. In *AAAI Fall Symposium on Genetic Programming*, 1995. (in press).

[10] Thomas Haynes, Roger Wainwright, and Sandip Sen. Evolving cooperation strategies. Technical Report UTULSA-MCS-94-10, The University of Tulsa, December 16, 1994.

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

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

[13] 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.

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

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

[16] Kenneth E. Kinnear, Jr. Alternatives in automatic function definition: A comparision of performance. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pages 119–141. MIT Press, Cambridge, MA, 1994.

[17] 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.

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

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

[20] 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.

[21] Mauro Manela and J. A. Campbell. 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, August 24-27 1993.

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

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

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

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

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

[27] 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.

[28] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[29] 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.

[30] 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.