Strongly Typed Genetic Programming in Evolving Cooperation Strategies

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

Abstract

A key concern in genetic programming (GP) is the size of the state–space which must be searched for large and complex problem domains. One method to reduce the state–space size is by using Strongly Typed Genetic Programming (STGP). We applied both GP and STGP to construct cooperation strategies to be used by multiple predator agents to pursue and capture a prey agent on a grid–world. This domain has been extensively studied in Distributed Artificial Intelligence (DAI) as an easy–to–describe but difficult–to–solve cooperation problem. The evolved programs from our systems are competitive with manually derived greedy algorithms. In particular the STGP paradigm evolved strategies in which the predators were able to achieve their goal without explicitly sensing the location of other predators or communicating with other predators. This is an improvement over previous research in this area. The results of our experiments indicate that STGP is able to evolve programs that perform significantly better than GP evolved programs. In addition, the programs generated by STGP were easier to understand.

1 Introduction

A problem with using Genetic Programming (GP) to solve large and complex problems is the considerable size of the state–space to be searched for generating good solutions. Even for small terminal and function sets and tree depths, search spaces of the order of $10^{30} - 10^{40}$ are not uncommon [Montana 1994]. To address this pressing problem, researchers have been investigating various means to reduce the GP state–space size for complex problems. Notable work in this area include Automatically Defined Functions (ADF) [Kinnear 1994b, Koza 1994], module acquisition (MA) [Angeline 1994, Kinnear 1994b], and Strongly Typed Genetic Programming (STGP) [Montana 1994]. The first two methods utilize function decomposition to reduce the state–space. The STGP method utilizes structuring of the GP S-expression to reduce the state–space.

We strongly agree with Montana's claim of the relative advantage of STGP over GP for complex problems [Montana 1994]. Besides the benefit of reducing the state–space, we are interested in whether the structure imposed by strong typing will be useful for analyzing the output of the evolved program. A common problem in AI research is deciphering the complex rules derived by the learning system. We believe that solutions produced by STGPs are in general more comprehensible than solutions produced by GPs.

In this paper, we further investigate the relative merits of STGP over GP by applying these methods on a difficult agent coordination problem. To our knowledge, this is the first application of the GP paradigm to the field of Distributed Artificial Intelligence (DAI). Our goal is to generate programs for the cooperation of autonomous agents in a simulated environment. The identification, design, and implementation of strategies for cooperation is a central research issue in the field of DAI. Researchers are especially interested in domains where multiple, autonomous agents share goals and resources, and use mutually acceptable work–sharing strategies to accomplish common goals. Developing cooperation strategies to share the work load is an extremely difficult problem, especially when the environment in which the agents are working is uncertain or not completely understood. Current techniques in

*This is a preprint of the paper in the *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995.

developing cooperation strategies are mostly done off–line using 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 cooperation strategy. In most cases a cooperation strategy is chosen if it is reasonably good.

In [Haynes 1994], we presented a new approach to developing cooperation strategies for multi–agent problem solving situations. Our approach differs 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., they are constructed 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.

The approach proposed in [Haynes 1994] for developing cooperation strategies for multi–agent problems is completely domain independent, and uses the GP strategy. To use the GP approach for evolving cooperation strategies, it is necessary to find an encoding of strategies depicted as S–expressions and choose an evaluation criterion for a strategy corresponding to an arbitrary S–expression. Populations of these structures are evaluated by a domain–specific evaluation criterion to develop, through repeated problem solving, increasingly efficient cooperation strategies. The mapping of various strategies to S–expressions and vice versa can be accomplished by a set of functions and terminals representing the fundamental actions in the domain of the application. Evaluations of 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.

We have used the predator–prey pursuit game to test our hypothesis that useful cooperation strategies can be evolved using the STGP paradigm for non–trivial problems. This domain involves multiple predator agents trying to capture a prey agent by surrounding it. The predator–prey problem has been widely used to test new coordination schemes [Gasser 1989, Stephens 1989, Stephens 1990, Korf 1992]. The problem is easy to describe, but extremely difficult to solve; the performance of even the best manually generated coordination strategies is less than satisfactory. We will show that STGP evolved coordination strategies perform competitively with the best available manually generated strategies. Our experiments demonstrate the relative advantage of using STGP over GP.

2 Strongly Typed Genetic Programming

Genetic programming (GP) is a powerful technique for automatically generating computer programs to perform a wide variety of tasks [Koza 1992]. The GP uses the traditional genetic algorithm (GA) operators for selection and recombination of individuals from one population of structures to form another population. The representation language used in GPs are computer programs represented as Lisp S–expressions in a parse tree. Recently GP has attracted a tremendous number of researchers because of the wide range of applicability of this paradigm, and the easily interpretable form of the solutions [Kinnear 1994a, Koza 1992, Koza 1994]. We assume the reader is familiar with the fundamentals of GAs and GPs.

In GP the user must specify all of the functions, variables and constants that can be used as nodes in a parse tree. Functions, variables and constants which require no arguments become the leaves of the parse trees and are called *terminals*. Functions which require arguments form the branches of the parse trees, and are called *non–terminals*. The set of all terminals is called the *terminal set*, and the set of all non–terminals is called the *non–terminal set*. Note the term *non–terminal* is what Koza [Koza 1992] calls a *function*.

One serious constraint on the user–defined terminals and non–terminals is called *closure*. Closure means that all of the non–terminals must accept arguments of a single data type (i.e. a float) and return values of the same data type. This means that all non–terminals return values that can be used as arguments for any other non–terminal. Hence, closure means any element can be a child node in a parse tree for any other element without having conflicting data types. Montana [Montana 1994] claims that closure is a serious limitation to genetic programming. Koza [Koza 1992] describes a way to relax the closure constraint using the concept of *constrained syntax structures*. Koza used tree generation routines which only generated legal trees. He also used operations on the parse trees which maintain legal syntactic structures. This is one of the fundamental concepts of STGP.

In STGP, 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. This causes the initialization process and the various genetic operations to only construct syntactically correct trees. One of the key concepts for STGP are generic functions, which is a mechanism for defining a class of functions, and defining generic data types for these functions. Generic functions eliminate the need to specify multiple functions which perform the same operation on different types. For example, one can specify a single generic function, VECTOR–ADD, that can handle vectors of different dimensions, instead of multiple functions to accommodate vectors for each dimension. Specifying a set of arguments types, and the resulting return type, for a generic function is called *instantiating* the generic function.

The STGP search space is the set of all legal parse trees. That is, all of the functions have the correct number of parameters of the correct type. Generally the parse tree is limited to some maximum depth. The maximum depth limit on a parse tree is one of the GP parameters. This keeps the search space finite and manageable. It also prevents trees from growing to an extremely large size.

Montana [Montana 1994] presented several different examples illustrating these concepts. He used STGP in solving a wide variety of moderately complex problems involving multiple data types. He showed in his examples that STGP was very effective in obtaining solutions to his problems compared to GP. Montana lists three advantages of STGP and generic functions:

1. Generic data types eliminate operations which are legal for some sets of data used to evaluate performance, but which are illegal for other possible sets of data.

2. When generic data types are used, the functions that are learned during the genetic programming process are generic functions.

3. STGP eliminates certain combinations of operations. Hence it necessarily reduces the size of the search space. In many cases the reduction is a significant factor.

In one of Montana's examples [Montana 1994], he presents a problem with a terminal set of size two, and a non–terminal set of size 10. When the maximum tree depth was restricted to five, the size of the search space for the STGP implementation was 10^5, while the size of the GP search space was 10^{19}. In the same example when the maximum tree depth was increased to six, the size of the search space for the STGP implementation was 10^{11}, while the size of the GP search space was 10^{38}.

3 The Pursuit Problem

The original version of the predator–prey pursuit problem was introduced by Benda, *et al.* [Benda 1985] and consisted of four blue (predator) agents trying to capture a red (prey) agent by surrounding it from four directions on a grid–world. This problem is a common domain used in Distributed Artificial Intelligence research to evaluate techniques for developing cooperation strategies. In the original version of the problem, agent movements were limited to one either horizontal or vertical step per time unit. The movement of the prey agent was random. No two agents (prey or predator) 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.

Gasser *et al.* [Gasser 1989] approached this problem by allowing the predators to occupy and maintain what is called a *Lieb configuration* while homing in on the prey. In a *Lieb configuration* each predator occupies a different quadrant, where a quadrant is defined by diagonals intersecting at the current location of the prey. This study did not provide any experimental results. Hence their research is difficult to compare with other work on this problem.

Korf [Korf 1992] claims in his research that a discretization of the continuous world that allows only horizontal and vertical movements is a poor approximation. He calls this the orthogonal game. Korf developed several greedy solutions to problems where eight predators are allowed to move orthogonally and diagonally. He calls this the diagonal game. He also developed solutions for a game in which six predators move on a hexagonal grid rather than a rectangular grid. He calls this the hexagonal 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 to solve all three types of games. The prey was captured in each of one thousand random configurations in these games. It should be noted that these games did not have a time limit, and once a prey was captured, it could not escape the predators.

Korf concludes that the *max norm* distance metric is suitable for the diagonal and the hexagonal game, but is ineffective for the orthogonal game. To improve the efficiency of capture (i.e., the steps taken for a capture), he adds a term to the evaluation of moves that requires predators to move away from each other before converging on the prey. Hence, the predators will encircle the prey and thus eliminate any escape routes. This measure is successful in the diagonal and hexagonal games, but makes the orthogonal game unsolvable. Korf replaces the traditional 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. It is our conjecture that the real difficulty is because in his experiments the predators and prey take turns moving. In all of our experiments the prey and predator agents move simultaneously.

4 Cooperation strategies

In our experiments, the initial configuration consisted of the prey in the center of the grid and the predators placed in random non–overlapping positions. The solutions we obtained are used to solve problems of other sizes, specifically grids of size 30 by 30, 40 by 40 and 50 by 50. Representative results are presented. All agents choose their action simultaneously. The environment is accordingly updated and the agents choose their next action based on the new state. Conflict resolution will be necessary since we do not allow two agents to co–occupy a position. If two agents try to move into the same location simultaneously, they are "bumped back" to their prior positions. One predator, however, can push another predator (but not the prey) if the latter did not move. The prey moves away from the nearest predator. However, 10% of the time the prey does not move. This effectively makes the predators travel faster than the prey. The grid is toroidal in nature, and the orthogonal form of the game is used. A predator can see the prey, but not other predators. Furthermore the predators do not possess any explicit communication skills, i.e. the predators cannot communicate to resolve conflicts or negotiate a capture strategy. We performed each of our experiments using both GP and STGP.

The STGP and GP algorithms are used to evolve a program to be used by a predator to choose its moves. The same program is used by all the predators. Thus, each program in the population represents a strategy for implicit cooperation to capture the prey. Further discussion of the evolution of these programs and comparisons of STGP versus GP is presented in Section 5.

4.1 Encoding of Cooperation Strategies

The terminal and function sets for our STGP implementation of the pursuit problem are shown in Table 1. 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: (*North*, *East*, *West*, *South* and *Here*). Notice the required types for each of the terminals, and the required arguments and return types for each function in the function set. Clearly, this is a STGP implementation for the pursuit problem.

4.2 Evaluation of Cooperation Strategies

To evolve cooperation strategies using GPs it is necessary to rate the effectiveness of cooperation strategies represented as programs or S–expressions. We evaluated each strategy by giving it k randomly generated predator placements. For each scenario, the strategy (program) was run for 100 time steps. This results in one simulation. A time step is defined as a move made by each of the agents simultaneously. The percentage of capture was used as a measure of fitness when comparing several strategies over the same scenario. Since the initial population of strategies are randomly generated, we expected that very few strategies would result in a capture after only 100 moves. Hence, we used additional terms in the fitness function to differentially evaluate the non–capture strategies. We designed our evaluation function for a given strategy 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. Higher fitness values result from strategies that bring the predators closer to the prey, and keep them near the prey. This term favors programs producing 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 (# 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.

- If a simulation ends in a capture position, an additional reward of (4 $*$ # of moves allowed $*$ grid width) is added to the fitness of the program. This term strongly biases the evolutionary search

Terminal	Type	Purpose	Function	Return	Arguments	Purpose/Return
B	Boolean	TRUE or FALSE.	CellOf	Cell	Agent A and Tack B	Get the cell coord of A in B.
Bi	Agent	The current predator.	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.)
Prey	Agent	The prey.	<	Boolean	Length A and Length B	If A < B, then TRUE else FALSE.
T	Tack	Random Tack in the range of Here to North to West.	MD	Length	Cell A and Cell B	Return the *Manhattan distance* between A and B.

Table 1: Terminal and Function Sets

toward programs that enable predators to maintain their positions when they capture the prey.

In our experiments, the distance between agents is measured by the *Manhattan distance* (sum of x and y offsets) between their locations.

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. Note these training cases can be either the same throughout all generations or randomly generated for each generation.

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

Program 1: The best program generated by GP.

5 Experimental Results

The STGP system, called GPengine, used in this research is an extension of the package developed in [Haynes 1995] and is written in C. Furthermore, it can be used as either a STGP or GP system depending on a runtime switch. A graphical reporting system was created for X–Windows using the Tcl and Tk toolkit with the Blt extension; this system was a modification of the work by Martin [Martin 1994].

The basic setup for the simulations is described in Section 3. Programs were evolved for grid sizes ranging from 10 by 10 to 50 by 50, with the prey either moving randomly (Random) or moving away from the nearest predator (MAFNP). In each generation k test cases were randomly generated, and each program was evaluated on each of the test cases. All fitness data presented is an average over the k test cases. Note that the fitness function is different per generation.

5.1 Untyped GP

Our results for the GP experiments is exemplified by the curve shown in Figure 1(a). Notice that there is no steady increase of fitness that indicates that learning is occurring.

The only good GP program resulted from a 20 by 20 grid with a randomly moving prey. The results for the experiment which produced the program is shown in Figure 1(b). The curve follows the pattern for good fitness curves which is indicative of learning. A good curve, in general, first exhibits a steady build up, then a sharp jump as a good building block is discovered, and finally reaches a plateau, which indicates the discovery of a local minimum/maximum. This process is repeated until either the global minimum/maximum is found or the GP terminates.

Note that the number of generations has been extended to 2000 in Figure 1(b) compared to 500 in Figure 1(a). This extension was prompted by promising results in the early generations. Generation 1389 had the best program (see Program 1) which contains 39 nodes and has a fitness of 29850 out of a maximum possible fitness of 32000.

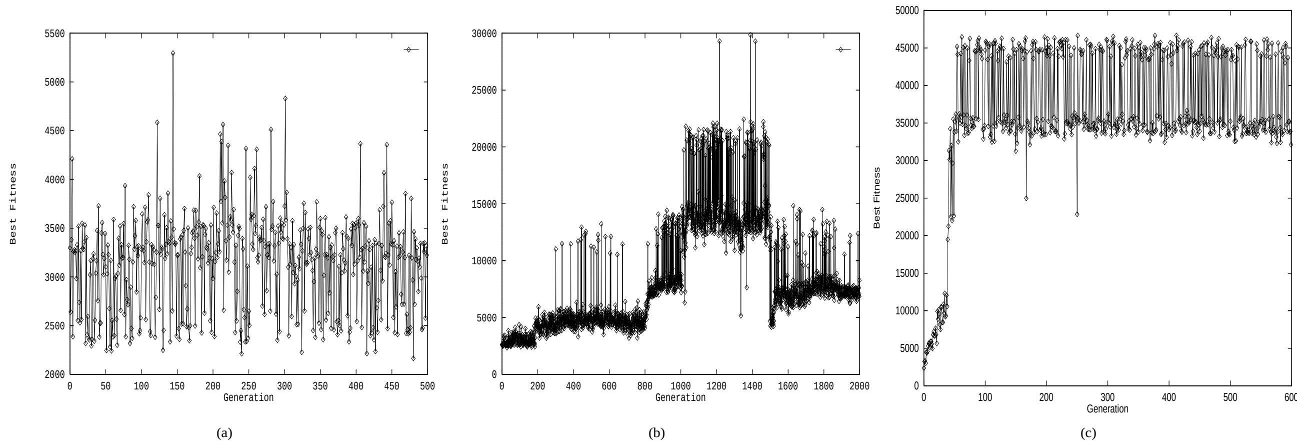

Figure 1: Example fitness curves for randomly moving prey systems: (a) Typical untyped (GP), (b) Best untyped (GP), and (c) Best typed (STGP).

5.2 Typed GP

A typical result for a STGP run using a 30 by 30 grid with a randomly moving prey is shown in Figure 1(c). This fitness curve indicates that good building blocks are being identified. This fitness curve learns faster than the GP results shown in Figure 1(b). Indeed *all* of our results for the STGP runs exhibited this property of learning.

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 average and maximum fitness of structures in successive populations (see Figure 1(c)). 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 2) which contains 61 nodes and has a fitness of 46660 out of a maximum possible fitness of 48000.

```
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 2: The best program generated by STGP.

5.3 Analysis

The moves suggested 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) shows the strategy for a deterministic Korf's *max norm*, a modification of the algorithm presented in [Korf 1992]. It is interesting to note how the agents converge on the prey using the policy. More significantly, 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. Predators are only assumed to be cognizant of the location of the prey and not the location of the other predators.

The moves suggested by the GP program are not shown due to the difficulty in the interpretation the program. The moves of the STGP program are only dependent on the relative relationship between a predator and the prey. The moves of the GP program are dependent on the actual relationship between both a predator and the prey and the fourth predator and the prey. In our description of the predator–prey domain, we stated that predators could not see each other. The closure property allowed the GP system to violate this rule. Also the STGP program's moves are independent of which particular predator is executing the program, while the GP program's moves are dependent on

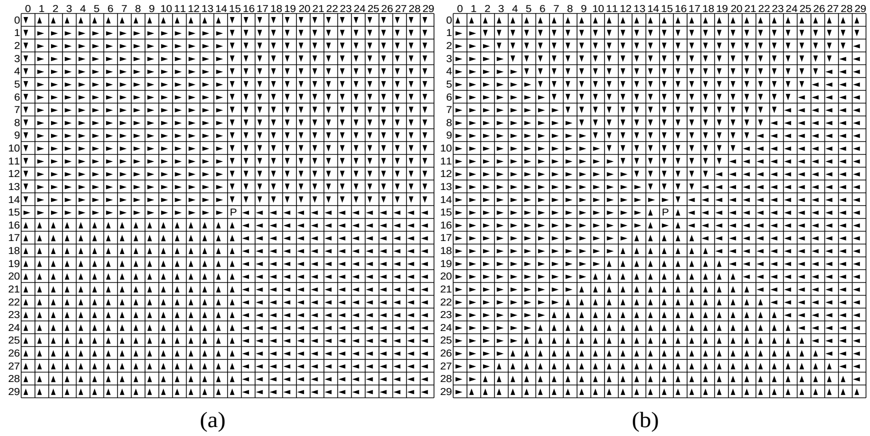

Figure 2: Example pursuit paths found by (a) STGP and (b) *max norm*.

a predator.[1]

The best GP program could be due to the random configuration of the predators rather than the result of a good program. Notice the three jumps in fitness in the generation range of 1200 to 1400 in Figure 1(b). The best STGP program is due to a good program, as evidenced by consistent scoring in Figure 1(c).

The best GP and STGP programs generated were tested against data from Stephens [Stephens 1989] and 1000 random test cases of our own. The averaged results are shown in Table 2. Four human derived algorithms, discussed in detail in [Haynes 1994], are also shown in Table 2: Korf's *max norm* (MN), Korf's *Manhattan distance* (MD), Korf's original *max norm* (MNO), and Korf's original *Manhattan distance* (MDO). The *max norm* algorithms determine the best move to make based on the diagonal distance between a predator and the prey. The *Manhattan distance* algorithms determine 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 predators in the original algorithms take turns moving, and thus have no conflict for cells. The predators in the modified algorithms all follow the rules outlined in Section 3.

Two lines of data are shown for both GP and MNO in Table 2. The first line represents the number of captures that hold to the end of a simulation. The second line represents *shadow captures*, which are situations in which the predators capture a prey, but allow it to escape. Results show that the GP program is weaker than the other algorithms in that it did not learn to keep the prey captured in all cases.

The relevant results extracted from Table 2 are:

- The STGP program significantly outperforms the GP program in all cases.
- While one manually derived algorithm, MD, consistently outperforms the STGP program, the STGP program was able to outperform all of the other greedy algorithms.
- The GP program did outperform some of the greedy algorithms.
- The GP program did not generalize outside of the domain that created it. This program was generated from a system that had a prey which moved randomly and also at the same time as the predators. The only significant capture rates for the GP program occur in this scenario.

There are two points that stand out when comparing GP versus STGP programs:

1. A good GP program takes longer to generate than a good STGP program.
2. The best STGP program had a higher capture rate than the best GP program.

Table 3 shows the state–space size for the traditional GP (untyped) implementation, and the STGP (typed) implementation of the pursuit problem. Notice for various maximum tree depth restrictions that the STGP has a significantly smaller state–space size. We believe this is probably the single most important reason for the observed performance difference.

[1]We have performed further experiments in which the STGP predators are allowed to interact with each other. Over time, this capability was bred out of the best of generation program.

	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	0.00(0.00) 0.00(0.00)	0.00(0.00) 0.00(0.00)	0.00(0.00) 0.00(0.00)	1.85(1.05) 1.85(1.05)	0.00(0.00) 0.00(0.00)	7.00(1.06) 7.00(1.06)	2.80(1.48) 2.80(1.48)	9.28(13.03) 9.31(13.30)
STGP	3.04(1.56)	5.04(2.11)	3.69(1.65)	9.50(2.25)	100.00(9.61)	216.00(8.12)	111.40(8.51)	340.20(14.69)
MN	0.69(0.62)	1.46(1.24)	1.92(1.16)	10.31(3.13)	28.70(5.36)	56.50(6.84)	68.00(8.11)	374.40(11.97)
MD	8.50(2.59)	5.31(2.41)	5.35(1.83)	13.35(2.37)	337.30(13.58)	222.90(12.33)	193.10(11.81)	463.40(19.727)
MNO	0.00(0.00) 0.39(0.64)	0.00(0.00) 0.58(0.70)	0.08(0.27) 7.19(2.37)	0.08(0.27) 7.23(2.37)	0.10(0.32) 14.90(4.31)	0.10(0.32) 15.00(5.29)	1.80(0.92) 273.20(16.79)	2.60(0.97) 270.80(17.76)
MDO	2.35(1.33)	2.27(1.59)	1.15(1.01)	1.04(1.15)	96.00(11.16)	96.70(11.89)	56.7(8.86)	57.30(7.73)

Table 2: Average number of captures over different runs (standard deviations are presented in parentheses).

Maximum Level	Traditional GP (untyped)	STGP (Typed)
0	9	5
1	972	50
2	9.5 x 10^8	30250
3	8.5 x 10^{26}	9.3 x 10^{11}
4	6.1 x 10^{80}	8.9 x 10^{32}
5	2.3 x 10^{242}	8.6 x 10^{92}

Table 3: State–Space Sizes for Various Configurations for the Pursuit Problem

In addition to reducing the state–space size, the STGP implementation generates programs which are easier for humans to understand. Our experimental setup explicitly excluded predators from sensing each others positions. The GP system found a loophole in that closure allowed the *CellOf* function to examine other predators. This occurred in Program 1 where each predator examined the relationship between itself and predator 4. This loophole complicates the process for determining the rules employed for movement.

6 Conclusions

We used Genetic Programming to evolve cooperation strategies for predators to capture a prey moving in a grid–world. Results from both the 30 test cases used in a previous study [Stephens 1989], and on an additional 1000 randomly generated test cases show that the solution evolved by the STGP implementation is very competitive with manually derived algorithms, and loses only to the MD algorithm. Furthermore, though the GP solution is significantly inferior to the STGP solution, it will occasionally fare better than the manually derived algorithms. The capture rates of all algorithms are still low, which indicates there is still a lot of work to be done in this domain.

We have shown that the STGP paradigm can be effectively used to generate complex cooperation strategies without being provided 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 should be.

7 Future Work

Have we packaged too much into the *MD* function? In constructing a GP system there are two efforts which are the most time consuming: deriving the function and terminal sets and constructing the fitness evaluation function. A concern in constructing the function set is how much functionality should be made available to the system? If there is not enough functionality, then a solution can not be found, and if there is too much functionality, then a solution is trivial.

In the STGP system the solution appears to be trivial, while in the GP system the solution is hard to find. If we replace the *MD* function with its constituent parts, will the STGP system be able to still find valid solutions and will it still fare better than the GP system?

Is the advantage of the STGP system due solely to the reduction of the state–space? If we were to increase the state–space of the STGP system to be equal or larger than the GP system then would the STGP system still perform better than the GP system? Simple calculations show that if we allow the STGP programs to grow two more depth levels, then its state–space will be significantly larger than the GP system.

In this paper we have utilized the significantly smaller search space for the STGP system over the GP system. We believe that this huge reduction in the search space allows the STGP system to evolve considerably better solutions. An important question to answer is whether the severely constrained search space for the STGP prevents it from generating optimal or near-optimal solutions in certain problems. This can happen if the reduced search space does not contain the optimal solution, or it eliminates portions of the search space that contain good building blocks required to construct an optimal or near-optimal solution.

We believe the above scenario does not arise in the problems discussed in the current paper. It is also not immediately obvious if such a scenario can occur in any problem domain. We would like to investigate this problem further. Our goal is to construct an artificial problem to show that a STGP system generated from a given GP system for that problem has a search space that does not contain the optimal solution.

Acknowledgments

This research was partially supported by OCAST Grant AR2-004 and Sun Microsystems, Inc.

References

[Angeline 1994] 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.

[Benda 1985] M. Benda, V. Jagannathan, and R. Dodhiawalla. On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center, Boeing Computer Services, Bellevue, WA, August 1985.

[Gasser 1989] Les Gasser, Nicolas Rouquette, Randall W. Hill, and John Lieb. Representing and using organizational knowledge in DAI systems. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*, pages 55–78. Pitman, 1989.

[Haynes 1994] Thomas Haynes, Roger Wainwright, and Sandip Sen. Evolving cooperation strategies. Technical Report No. UTULSA-MCS-94-10, The University of Tulsa, December 16, 1994.

[Haynes 1995] 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.

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

[Kinnear 1994b] 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.

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

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

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

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

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

[Stephens 1989] Larry M. Stephens and Matthias B. Merx. Agent organization as an effector of dai system performance. In *Working Papers of the 9th International Workshop on Distributed Artificial Intelligence*, September 1989.

[Stephens 1990] Larry M. Stephens and Matthias B. Merx. The effect of agent control strategy on the performance of a DAI pursuit problem. In *Proceedings of the 1990 Distributed AI Workshop*, October 1990.