Evolving Agents

Adil Qureshi
Genetic Programming Group,
Dept. of Computer Science,
University College London,
Gower Street,
London WC1E 6BT, UK
Email: A.Qureshi@cs.ucl.ac.uk
URL: http://www.cs.ucl.ac.uk/staff/A.Qureshi/

Abstract

The paradigm of *agent based computing* is becoming increasingly popular both in *distributed artificial intelligence* and as a general software engineering technique. The difficulty with agent based computing is that success depends not just on the correctness of any one agent, but on the *emergent behaviour* arising from the interaction of a *society* of agents. As a consequence, the problem of programming agents is non trivial and poorly understood.

In this paper we show that genetic programming can be used to automatically program agents which communicate and interact to solve problems. The programs evolved simultaneously define when and what to communicate, and how to use the communicated information to solve the given problem.

1 Introduction

In previous work GP approaches have concentrated on evolving *reactive agents*. The latter work on the principle that complex intelligent behaviour can emerge from the interaction of a collection of very simple agents. These agents obtain knowledge about actions and behaviours of other agents by sensing changes in the environment. Any communication is at a very low level, achieved implicitly by making changes to the environment rather than explicitly by exchanging messages with each other. Koza used GP to evolve reactive agents for the central food foraging problem [5]. Haynes evolved agents for the predator/prey (capture) problem [3], and others [2], [4]. Reynolds [8] co-evolved agents to simulate the game of tag. Manela [7] employed GAs to optimise architectural features of agents (including simple communication) for the pursuit problem. Andre [1] used GP to co-evolve program pairs, the first of which creates a map of the environment, and the second of which uses this map to control a mobile robot. The programs are run in sequence (writer first, then reader) and can be considered to be using a form of unidirectional communication.

In this paper we concentrate on using GP to evolve agents that communicate directly with each other, via message passing. These evolved agents must decide what information is to be communicated and how it is to be used to solve the global problem.

2 The Two Agents Problem

The goal of our experiments is to see if GP can be used to evolve agents that communicate to solve problems. For this purpose we have defined a simple problem for which it is necessary for communication to take place and furthermore, for which it is easy to measure the effectiveness of any communication. The problem involves two agents A and B that steer two vehicles placed on a two dimensional grid. The vehicles are constantly moving in the direction that they are currently pointing. All movement is discrete, so that the vehicles always have integer (x,y) coordinates. Initially the vehicles are pointing north. The steering mechanism allows the orientation of the vehicles to be changed to any one of the 8 compass directions. The goal is to evolve the code for agents A and B such that they steer the vehicles to enable them to meet. The vehicle model closely follows that used by Reynolds [8].

The vehicles are given energy proportional to the euclidian distance between them. The latter is an attempt to prevent the "meet you at (x,y)" strategy in which the agents obviate any need for communication having previously agreed to always move to the fixed position (x,y). A single energy unit is consumed each time a vehicle moves one step in any direction. The game ends when both vehicles run out of energy or when they meet.

Agents A and B know only their own positions, but both have access to two bi-directional unbuffered communication channels called ChannelA and ChannelB which can be used to send and receive messages to each other. Each communication channel is "wired" so that one end is connected to one vehicle and the other to the other vehicle. A message sent from one end can only be received at the other end, and vice versa.

The communication channels have been designed so they can only be used to send and receive messages in the form of integers. When an attempt is made to read from a channel the last message received is delivered (default value 0).

Clearly what we would like is for agents A and B to evolve behaviours that attempt to use the communication channels to exchange position information.

The success of the agents is measured using the following fitness function:-

$$Fitness(A) = Fitness(B) = \frac{D_{before}(A,B)}{D_{before}(A,B) + D_{after}(A,B)}$$

Where D_{before} and D_{after} are the distances between A and B before and after execution respectively. This function produces a range of fitness values varying between 0 and 1. A value of 0.5 indicates that the agents remained the same distance apart. Values below 0.5 indicate that the agents moved further apart. The value 0 is actually not possible since it implies that the agents were at an infinite distance apart when they ran out of energy. Values above 0.5 indicate that the agents moved closer together, and a value of 1 indicates that they met.

Note that for the agents to meet, we actually mean that they must be at the same coordinates at the same time. Since the vehicles are constantly moving, the latter implies synchronised movement which makes the problem more difficult.

2.1 Architecture

Tables 1 and 2 provide details of the architecture used to evolve programs to steer the agents. Note that all the primitives are integer based. The program makes use of a main branch and two ADFs [6]. ADF0 is a parameterised function and takes two arguments. Its function and terminal set do not contain any environment sensing or effecting facilities. All communication and most environment sensing facilities are available only to ADF1. The main branch has only ADF0 in its function set. The terminal set includes ADF1 and GetD, where the latter is the only environment sensing primitive not available to ADF1. The idea is that ADF0 is a computational unit and ADF1 is a communication unit.

A fixed set of 25 randomly generated initial positions (within the bounded region (-100,-100) to (100,100)) were used to evaluate each individual in the population. The raw fitness values for each of the 25 cases were averaged to arrive at an overall fitness value for the individual. The energy allocated to each individual was $1.5x$ the initial distance between the agents. This provides sufficient energy to allow solutions that don't use diagonals to evolve, but at the same time inhibits solutions that use the "meet you at (x,y)" strategy.

Each individual in the population was evaluated by using it to steer both A and B. This is achieved by alternatingly executing the GP once in the context of A, and then in the context of B. The result of each execution is used to steer the respective vehicle, which is then moved forward one unit. This process is repeated until either the two agents both run out of energy, or they meet. The pseudo code in table 3 illustrates this approach.

Table 3: Evaluation Pseudo Code

```
agent = A
while ((A.energy || B.energy) && (!met(A,B))) {
  if (agent.energy) {
    steering_delta = eval GP in context of agent
    agent.turn(steering_delta)
    agent.move()
    agent.energy--
  }
  if (agent == A) agent = B else agent = A
}
```

The turn() function adds the steering delta to the current direction. Negative delta values can be used to steer the vehicle anti-clockwise, and positive values clockwise. The turn() function guarantees that any resulting direction is always in the range 0-7.

Additional code was added to the move() function so that when enabled, it generates output for the LOGO turtle graphics language. This output provides a very simple means of visualising the performance of a GP for each of the test cases.

After the completion of a run, the best individual was re-evaluated using 1000 test cases instead of twenty five to check for the generality of the solutions. A further re-evaluation used the original 25 test cases, but with send operations disabled on both communication channels for both agents. The latter is used to provide a measure of the dependency of the solution on communication.

3 Results

The results for the 3 best out of 10 different runs are shown in table 4. Clearly the best individual is from run 7, which is also the most general solution. As expected, all three individuals score very low fitness values when both of the communications channels are disabled.

LOGO output tracing in "realtime" the movement of the agents, together with the UCB LOGO package can be obtained from the directory "/genetic/agents2" at the ftp site "cs.ucl.ac.uk" at UCL. The file named README explains how to view the results. Figure 1 shows snapshots of the output trace for the best individuals from the three best runs for sample test cases. The snapshots were taken at the point when the agents met or when

Table 1: Actions Performed by Terminals and Functions

Primitive	Purpose
`Add`(x,y)	**return** $x+y$
`Sub`(x,y)	**return** $x-y$
`Mlt`(x,y)	**return** $x*y$
`Div`(x,y)	**return** x/y
`IfGT`(x,y,$t1$,$t2$)	**if** $x > y$ **return** $t1$ **else return** $t2$
`RecvA`	**if** no message received yet at *ChannelA*, **return** 0; **else return** last message received
`RecvB`	**if** no message received yet at *ChannelB*, **return** 0; **else return** last message received
`SendA`(x)	send x down *ChannelA*; **return** x
`SendB`(x)	send x down *ChannelB*; **return** x
`GetX`	get the current X coordinate; **return** X
`GetY`	get the current Y coordinate; **return** Y
`GetD`	get the current direction D; **return** D
`Arg1, Arg2`	arguments of current ADF
`Zero`	constant value 0; **return** 0

Table 2: The Two Agents Problem

Objective:	Find a program that steers the two agents so that they meet before they run out of energy.	
Primitives:	Functions	Terminals
`Main`	`Adf0`	`Adf1, GetD`
`Adf0`	`Add, Sub, Mlt, Div, IfGT`	`Arg0, Arg1, Zero`
`Adf1`	`SendA, SendB, Add, Sub, Mlt, Div, IfGT`	`RecvA, RecvB, GetX, GetY, Zero`
Fitness cases:	25 randomly generated fixed test cases.	
Energy:	$1.5 * Distance_{before}(A, B)$	
Raw Fitness:	$Fitness = \frac{Distance_{before}(A,B)}{Distance_{before}(A,B)+Distance_{after}(A,B)}$	
Fitness:	Average of the raw fitness scores of the 25 test cases times 10,000.	
Parameters:	Pop = 1000, G = 100, one dimensional demes (size 25), grid range (-100,-100) to (100,100).	
Termination Condition:	A program gets a fitness score of 10,000 or current generation exceeds G.	

they ran out of energy.

Table 4: Results

Run	Length	Fitness (25)	Fitness (25) No Comms	Fitness (1000)
0	74	8028	5081	6638
2	163	8122	4710	6300
7	211	8831	4996	8979

3.1 Run 0

The best individual from run 0 (table 5) achieves high fitness for the training test cases, but scores a considerably lower fitness value when faced with the more extensive unseen test cases. The strategy used by this individual is not very clearly defined.

X coordinates are exchanged using ChannelB, and the agent with the greater X value moves diagonally in the North-West direction until X coordinates are equalised. No clear strategy is apparent for equalising Y coordinates, however the agent which starts off with the greater X value never moves South, hence meetings are only be achieved when the other agent moves North.

Table 5: Best Individual from Run 0

Main: ((ADF0 ADF1 GetD))
ADF0: ((IfGT (Mlt Arg1 Arg1) (Add Arg0 Arg1) (Add (Div Arg1 Arg1) (Add Arg0 Arg1)) (Add (Mlt Arg1 Arg1) (Mlt (Mlt Arg1 Arg1) (IfGT Arg0 Arg0 Arg1 Arg0))))))
ADF1: ((Mlt (Mlt (IfGT GetX RecvB RecvA RecvB) (Mlt (Mlt (IfGT GetX RecvB RecvA RecvB) (IfGT GetX RecvB RecvA RecvB)) (SendB GetX))) (Mlt (Mlt (IfGT GetX RecvB RecvA RecvB) (Mlt (Mlt (IfGT GetX RecvB RecvA RecvB) (IfGT GetX RecvB RecvA RecvB)) (SendB GetX))) (SendB GetX))))
Fitness: 8028
Structural Complexity: 74

3.2 Run 2

The best individual from run 2 (table 6) again achieves high fitness for the training tests, but again is significantly less successful with the more extensive unseen test cases. The strategy used by this individual is also not very clear.

This time Y coordinates are exchanged using ChannelA, and the agent with the lower Y coordinate moves North until Y coordinates are matched. If both agents have negative Y coordinates, they both move North until they reach the line Y = 0. The agent with the higher Y coordinate (or the first to reach Y = 0 for the negative Y coordinates case) moves left or right in semi-circles. This makes it difficult for the other agent to match its Y coordinate exactly. The result is that when they are at approximately the same Y coordinates, they both end up following the cyclic movement. If during this movement, they synchronise Y values, they remove the vertical component from their movement and follow an East or West path. If they end up pointing towards each other, the latter facilitates a meeting.

Table 6: Best Individual from Run 2

Main: ((ADF0 ADF1 GetD))
ADF0: ((IfGT Zero Arg0 Arg1 (IfGT Zero Arg0 Arg1 Arg0)))
ADF1: ((IfGT (Div (Mlt RecvA Zero) (Mlt RecvA Zero)) (Div (Div RecvA RecvA) (Sub GetY RecvA)) (IfGT Zero Zero Zero RecvA) (IfGT (SendA GetY) (Sub GetY RecvA) (IfGT (Div (IfGT RecvB Zero GetY RecvA) (Sub GetY GetX)) (Sub (IfGT RecvB Zero GetY RecvA) (Sub GetY RecvA)) (Add (SendA GetY) (Sub (Sub GetY RecvA) RecvA)) (IfGT (Div RecvA RecvA) (Sub GetY RecvA) (IfGT (Div (IfGT RecvB Zero GetY RecvA) (Sub GetY GetX)) (Sub GetY RecvA) (IfGT (SendA GetY) (SendA (IfGT (SendA GetY) (Sub GetY RecvA) (IfGT (Div (IfGT RecvB Zero GetY RecvA) (SendA GetY)) (Div RecvA RecvA) (Sub (Div RecvA RecvA) (Sub GetY RecvA)) (IfGT (Div (SendA GetY) (Sub GetY GetX)) (SendA GetY) (Div (IfGT RecvB Zero GetY RecvA) (Sub GetY GetX)) (SendA GetY))) (SendA GetY))) (IfGT Zero Zero Zero RecvA) (Sub GetY RecvA)) (Div (IfGT RecvB Zero GetY RecvA) (Div (IfGT RecvB Zero GetY RecvA) (Sub GetY RecvA)))) (SendA GetY))) (SendA GetY))))
Fitness: 8122
Structural Complexity: 163

3.3 Run 7

The best individual for run 7 (table 7) not only achieves the highest fitness value, it also proves to be the most general solution when tested over 1000 different tests, scoring an even higher fitness value. The strategy used by this individual is very well defined.

The basic strategy is to equalise Y coordinates by making use of communication channel A. X-coordinates are not exchanged, so instead the agents have "agreed" to meet at the ordinate X = -1. This works best if the agents are on opposite sides of the line X = -1 or very close to this line. If however the agents are on the same side of the line X = -1, but distant from it, they waste energy by moving without getting any closer to each other, and are hence more likely to run out of energy before they meet.

The behaviour of an agent at any instant, can be approximated by the pseudo code in table 8.

Table 7: Best Individual from Run 7

Main: ((ADF0 GetD ADF1))
ADF0: ((Add (Sub Arg1 Zero) (Add (Mlt (Sub Arg1 Zero) (Sub Arg1 Zero)) (IfGT Arg0 Arg0 Arg0 Arg0))))
ADF1: ((IfGT (SendA GetY) (Div (SendA RecvA) (IfGT (Add GetX Zero) (SendB RecvA) (Add GetX Zero) (Div GetX Zero))) (Add (Div (IfGT RecvA RecvA Zero GetY) (IfGT (IfGT (SendB RecvA) (Sub (IfGT RecvA RecvA Zero GetY) (SendB RecvA)) (Div GetX Zero) (Add GetX Zero)) (Add GetX Zero) (Add (SendB RecvA) (Div GetX Zero)) (IfGT (SendB RecvA) (IfGT (IfGT (SendB RecvA) (IfGT RecvA RecvA Zero GetY) (Add GetX Zero) (IfGT RecvA RecvA Zero GetY)) (Sub (IfGT RecvA RecvA Zero GetY) (SendB RecvA)) (Div GetX Zero) (Add GetX Zero)) (IfGT (SendB RecvA) (Add (Div (IfGT RecvA RecvA Zero GetY) (IfGT (Div GetX Zero) (Add GetX Zero) (Add (SendB RecvA) (Div GetX Zero)) (IfGT (SendB RecvA) (IfGT (IfGT (SendB RecvA) (Div (IfGT RecvA RecvA Zero GetY) (SendB RecvA)) (SendB RecvA) (IfGT RecvA RecvA Zero GetY)) (Sub (IfGT RecvA RecvA Zero GetY) (SendB RecvA)) (Div GetX Zero) (Div (SendA GetY) (SendB RecvA))) (IfGT (SendB RecvA) (Add (Div GetX Zero) (SendB RecvA)) (Div GetX Zero) (Add GetX Zero)) (SendB RecvA)))) (Div (IfGT RecvA RecvA Zero GetY) (IfGT RecvA RecvA Zero GetY))) (Div GetX Zero) (Add GetX Zero)) (SendB RecvA)))) (Div (SendA GetY) (SendB RecvA))) (Div GetX Zero)))
Fitness: 8831
Structural Complexity: 211

Table 8: Strategy of Best Individual from Run 7

```
if (my Y > 0) and (my Y >= remote Y)
    Send my Y
    if (my X > -1)
        move East
    else
        move West
else if (my Y > 0) and (my Y < remote Y)
    Send my Y
    move North
else if (my Y < 0)
    Send 2
    move North
```

4 Conclusion

We have demonstrated that it is possible to evolve agents that communicate with each other to solve a global problem. In doing so, the agents effectively decide what to communicate and when to communicate it. They then use the information communicated to solve the global problem. In summary behaviour directed by communication automatically defines who does what and when.

In the experiments performed the code generated only used one of the communication channels, for exchanging position information pertaining to only one of the axis. All the runs used heuristics to guide movement along the axis for which no information was exchanged. This problem could be due to any number of reasons, including the chosen architecture, loss of population diversity, or even the difficulty of the problem. Early analysis suggests that the problems are related to population diversity, but this will need to be investigated further. The results do however clearly show that GP programs can use communication to exchange information vital to solving a problem.

The long term goal of our project is to see if agents that use a very primitive language (for example, consisting of just verbs and nouns) to communicate and solve complex problems can be evolved.

5 Future Work

5.1 Extending the Two Agent Problem

Currently we do not take into account the total units of energy spent during the test cases. Hence agents that are more wasteful can potentially end up with the same fitness score as efficient ones. We will therefore use energy remaining as tie breaker when two individuals have the same fitness value.

It would be interesting to look at the effect of noisy communication channels, to see if noise tolerant communication protocols can be evolved. This will probably require the set of primitives to include indexed memory.

The communication channels are currently unbuffered, we could make them buffered, and furthermore add some cost so that each time a message is sent or received, energy is consumed. The amount of energy consumed would probably need to be much less than the cost of movement.

5.2 The Three Agent Problem

We are currently extending the two agent problem to a new variation of the pursuit problem. A third agent P is to be co-evolved, whose goal is to intercept A or B before they meet. P is equipped with communications facilities that allow messages to be received from and sent to both of the communication channels of A and B.

Experiments will be performed in passive and active phases. In the passive phase P will only be allowed listen to the communication channels connecting A and B. In the active phase, P will also be able to send messages to both A and B. The passive phase allows P to eavesdrop and hence obtain position information which should help it to intercept one of the agents. Agents A and B must hence use some form of encryption to prevent discovery. The active phase allows P to send decoy messages which would not be easy to distinguish between the real messages sent between A and B. This time A and B would need to develop methods of authentication. Clearly the set primitives would need to be extended to include indexed memory to facilitate the latter. In addition, a buffered communication model will probably be required.

Acknowledgements

A. Qureshi is funded by the EPSRC. I would like to thank my supervisors (Jon Crowcroft and Steve Hailes), and Adam Fraser for GPC++.

References

[1] David Andre. Evolution of mapmaking ability: Strategies for the evolution of learning, planning, and memory using genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 250–255. IEEE Press, June 1994.

[2] Thomas Haynes, Sandip Sen, Dale Schoenefeld, and Roger Wainwright. Evolving a team. In E. S. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 23–30, 445 Burgess Drive, Menlo Park, CA 94025, USA, 10–12 November 1995. AAAI.

[3] Thomas Haynes, Roger Wainwright, Sandip Sen, and Dale Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 271–278, San Francisco, CA, USA, July 1995. Morgan Kaufmann.

[4] Thomas D. Haynes and Roger L. Wainwright. A simulation of adaptive agents in hostile environment. In K. M. George, Janice H. Carroll, Ed Deaton, Dave Oppenheim, and Jim Hightower, editors, *Proceedings of the 1995 ACM Symposium on Applied Computing*, pages 318–323, Nashville, USA, 1995. ACM Press.

[5] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[6] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, May 1994.

[7] Mauro Manela. *Contributions to the Theory and Applications of Genetic Algorithms.* PhD thesis, University College, London, December 1993.

[8] Craig W. Reynolds. Competition, coevolution and the game of tag. In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 59–69, Cambridge, MA, USA, 1994. MIT Press.