The Three-Headed Monster: An Evolutionary Program

Dana Vrajitoru and Charles Guse

TR-20081217-1

Department of Computer and Information Sciences

Indiana University South Bend

Abstract

This report presents the model we used for our entry in the GECCO 2008 Contest: Finding a Balanced Diet in Fractal World [1]. The challenge consisted in developing an intelligent agent exploring a world for which it only knows the height in a local neighborhood, looking for a balance diet of game and grain. We used genetic programming to evolve our agent, using both a simple exploratory approach and more sophisticated search techniques. Our results show that in this particular context, the simple approaches seem to work the best.

1. Introduction

The challenge consisted in developing an agent moving in a two dimensional world containing food of two types: grain and game, with no prior knowledge of its location, and with the height map as the only indication of the state of the world. There is an implicit probabilistic correlation between the height of the terrain and the sort of food that can be found there. The agent's working information is the height of the terrain in a local vicinity, but there is no restriction about its memory. The agent had a given number of steps to find food, and was rewarded for the amount of both grain and game that it could find within those steps.

We developed an agent using genetic programming in three phases. In the first phase, we aimed to evolve a function exploring as wide a region as it could without repetition. In the second phase, we evolved the agent in the real contest conditions, and using the best function evolved during the first stage. In the third phase we introduced a coevolutionary approach where three populations of functions were evolved at the same time, serving as the test, positive action, and negative action in a conditional.

Overall our best agents managed to eat between 5% and 10% of the food on the map in a fairly balanced diet, and more sophisticated approaches didn't seem to yield the expected improvement in performance.

2. The Problem and Contest Rules

The goal of the competition was to evolve an agent to search a two dimensional landscape and find as much as possible of two types of food. The landscape consisted of a 256x256 grid. Each cell had an 'elevation' between 0 and 255. Figure 1 shows an example of such a map, where the cells are shaded by elevation, brighter colors meaning lower values. Two types of food are in this landscape: grain - green dots, and game - gold dots. In addition, red squares represent impassible terrain.

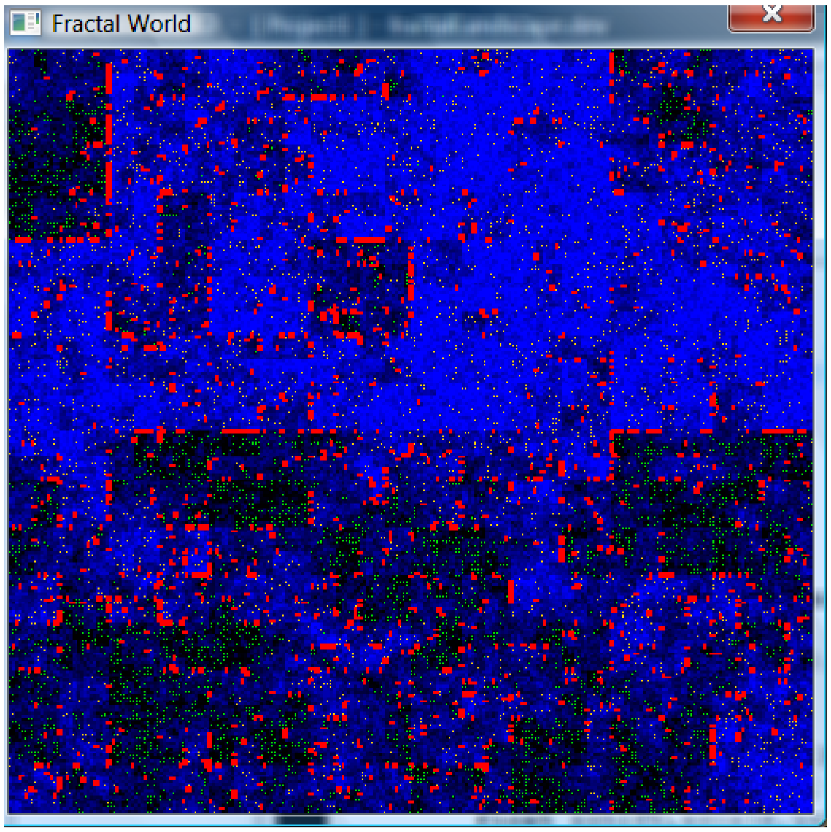

Figure 1. Sample map, brightness defines elevation, green dots represent grain, gold dots represent game, red represents impassable areas.

The agent's goal was to find a balanced mixture of grain and game. Specifically, for the competition an individual's fitness was evaluated as the amount of whichever food it found less of. For example, if an agent found 35 grain and 29 game, its score was 29. Individuals were allowed 13,107 moves, representing 20% of the surface of the terrain. Thus, exhaustive search algorithms were not expected to be particularly effective.

The agent only had information about the elevation of the terrain in a vicinity of the current position. There was an implicit probabilistic correlation though between the type of food more likely to be present at a particular location. Some statistics of the provided maps are shown in Table 1 below. From this table we can see that game can be found mostly at low altitude elevations, while grain can be found mostly at high altitude elevations.

Table 1. Distribution of game and grain based on elevation

	Game				Grain			
map	**Ave**	**StdDev**	**min**	**max**	**Ave**	**StdDev**	**min**	**max**
1	79.81	28.3	0	188	223.92	10.49	43	255
2	75.41	27.48	0	193	220.88	17.75	28	255
3	77.24	25.98	0	198	221.17	17.61	33	255
4	73.37	26.83	0	204	221.17	18.29	36	255
5	74.56	25.93	0	219	198.48	46.68	28	255
6	72.42	26.71	0	204	223.26	15.64	43	255
7	73.32	25.97	0	187	176.27	62.39	28	255
8	66.8	26.98	0	166	177.66	65.35	28	255
9	74.17	25.84	0	178	222.76	15.29	34	255
10	69.83	26.36	0	195	221.54	22.99	31	255
Average	73.69	26.64	0	193.2	210.71	29.25	33.2	255

Individuals could 'see' elevations in a 5 by 5 grid around their location, numbered as shown in Figure 2.. They automatically picked up food if they entered a square containing it, but could not see it. Thus, individuals had to learn where to look based on elevation. At each move, the current position of the agent is in cell 12, in the middle.

0	1	2	3	4
5	6	7	8	9
10	11	**12**	13	14
15	16	17	18	19
20	21	22	23	24

Figure 2. Visible neighborhood of the agent and its cell numbering

Individuals started on a random cell of unknown location to them. They must develop an ANSI C function with the prototype

```
int Next_Move(int *grid, int grain, int game, int last, int time);
```

returning the next relative position of the agent as a number between 0 and 24, as stated in Figure 2. The agent receives the local neighborhood for the next move as input for this function, as well as the last move it has executed before. Moves resulting in an impassible position, remaining in place, or attempting to move further away than the 5x5 grid resulted in the agent's position not changing, but still counted towards their total number of moves.

3. Our Solution and Implementation Details

There was once a little yellow goblin who was looking food. On his journey he encountered a wizard who turned him into a two-headed phase shifter. The wizard warned him that his health would be in danger if he didn't insure a balanced diet of game and grain.

Our initial plan was to divide the process in two phases: for the first half of the given steps we would develop a greedy algorithm trying to get as much food of any sort as it can. In the second half we would employ another algorithm attempting to improve on whichever food had been collected less so far.

As we were not sure what algorithms we could use early on, we mostly focused on genetic programming (GP) [2] with as little input from the human as we could. We used a small free Genetic Program written in Python by Paras Chopra [3] and modified for our purposes. We converted the resulting trees to C by hand.

3.1 First Phase

The phase shifting goblin was very hungry so he ran around for a while almost purposelessly looking for any and every food he could get his hands on.

For the first phase we started with a function evolved with a classic GP with the aim of exploring a square area trying to avoid walking over the same cells more than once. The chromosomes are trees with the following specifications, where the functions have the usual meaning that they have in any programming language, wrapped in some tests for mathematical soundness of the arguments:

Functions: +, -, *, /, %, cos, sqrt, pow

Variables: x and y

Constants: the range of possible moves, [0,24], plus the size of the table and some randomly generated constants

Output: a value capped to the range [0,24] representing the next move.

Other parameters: population size: 100, number of generations: 500, maximum tree depth: 30, selection: fitness-proportionate and elitist, probability of crossover: 0.8, probability of mutation: 0.1

These other parameters were kept for all the experiments, unless otherwise specified. We initially tried a larger number of generations but the population seems to converge rather quickly and it wasn't of much use letting it run a lot longer.

Fitness: we started with a table of 25x25 which was initially composed of unmarked cells. We tested each program by making 150 moves based on their output, updating the variables x and y after each move accordingly, and marking the visited cell each time. The fitness consists of the number of marked cells at the end of the run.

Figure 3 shows an example of the output from the program, corresponding to the tree shown to its right:

```
+--ADD
| +--SQRT
| | +--ADD
| | | +--[X]
| | | +--SQRT
| | | | +--ADD
| | | | | +--[Y]
| | | | | +--[9]
| +--SQRT
| | +--DIV
| | | +--[X]
| | | +--[20]
```

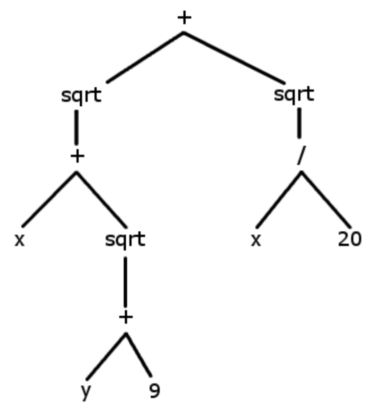

Figure 3. Output of the program (left) and corresponding tree (right)

The best program we obtained (see Appendix 1 for the full program output) achieved a fitness of 86 and was produced at generation number 162. We translated this output into a C function called walk_move. Interestingly enough, many of the functions we obtained this way seem to want to go constantly in one given direction.

We tested the function walk_move by itself in the real test conditions. As we didn't know the real position of the agent on the map, we started with random values for the variables x and y for which we used static variables, and updated these values based on the moves made. We treated the map as being circular, which means that 256 = 0 for both coordinates. This test revealed that the function has an eigenvalue and the agent eventually converges to that position on the map sooner or later. A solution to avoid this convergence is to re-scramble the variables x and y from time to time.

3.2 Second Phase

After getting full on food, the goblin shifted to a new phase where he was supposed to be more intelligent, but most of the time he was just standing still thinking that the food would come to him.

For the second phase we introduced the real test conditions into the fitness function. The size of the table is now 256 and the food and elevation tables are imported from one of the files that were provided. The fitness function is running 150 steps, but this time they are done in the table of size 256x256 and the game and grain points are counted. The fitness returns the least of game and grain.

The set of functions was extended to include the function walk_move that was evolved in the first phase, and also a function elev that returns the elevation at the position given by the argument in the 5x5 grid, if the argument is in the range [0,24], and -1 otherwise.

We evolved two functions this way. For the first one we added the following set of logical and comparison functions: {IF, EQ, NOT, GT, LT, AND, OR, XOR}. The best function evolved this way is translated into the function **game_walk**. As the logical and comparison function didn't seem to be well used, we removed them, and the best function evolved this way translated into the function **walk_elev_grain**. The original trees for both these functions can be found in the appendix. Neither of these two functions performed any better than just the function walk_move by itself so we decided not to use them. In fact they both seem to return a constant move most of the time.

3.3 Third Phase: The 3-Headed Giant

The wizard took pity on the goblin and summoned terrible forces that transformed the creature into a 3-headed giant. The bigger of the three heads was shouting commands to the two others, but after a while it became a tug a war between the two lower heads which caused the giant to just move back and forth in place.

For the third phase we tried a cooperative co-evolutionary approach where we evolved three populations in parallel. The first population evolves a condition using the set of logical and comparison functions introduced above. The second population evolves a function to be called if the condition is true. This population is not using the logical and comparison functions, but the set of arithmetic functions. We decided at this point to replace the function elev with just a variable containing the elevation at the current position in the grid. The third population evolves a function to be called if the condition is false, and its parameters are identical to the second population.

The fitness function is very similar to the one described for the second phase, except that it uses one chromosome from each of the 3 populations together to determine the move. The method for combining the 3 chromosomes is the following: for each generation, the chromosomes of the first population are evaluated using the best chromosome from the second and third populations from the previous generation. The second and third populations are evaluated in a similar way.

The best combined functions that we obtained with this new approach are shown in the Appendix 4 under the name **three_heads**. The performance of these functions in the real test conditions was quite inferior to the first function that was evolved in phase one.

3.4 Final Decision

The three headed giant exulted with so much arrogance that he turned into a supernova, which collapsed onto itself and became a red dwarf. The little red dwarf can still be heard roaming the fields full of game and grain, sadly mourning his lost glory.

Given that the more complex functions that we tried to evolve performed below our expectations, we decided to turn in the very first function walk_move that was evolved without any knowledge of the elevation and of the accumulated food.

As a wrapper around this function, we use a couple of static variables x and y that we initialize with random values in the range [0, 255]. We update these variables based on the current move and we enclose it in a loop that makes sure that the move returned is not 12 (stay in place) or a move to a position containing a block.

On the average we estimated that the algorithm collects a total amount of food between 5% and 10%, which is relatively well balanced between grain and game.

4. Conclusions

In this report we presented an evolutionary approach to developing an agent capable of exploring a two dimensional world in search for food in the specific conditions for the GECCO 2008 contest.

We introduced several models, the first one doing a simple exploration where the goal was to avoid repetition as much as possible, with no knowledge of the specific problem conditions. The second model we developed evolved under the real test conditions, where the fitness function used in the evolutionary process reflected the performance measure used in the contest. The third model introduced a co-evolutionary approach where three populations of genetic trees evolved in parallel, depending on each other for the fitness.

The results from the various tests indicate that the approach that performed best was the simplest one, not taking into consideration any information concerning the elevation and the food acquired so far. This agent was capable of acquiring about 5% to 10% of the available food in a number of moves covering at most 20% of the terrain, which is an encouraging result.

The GECCO 2008 contest was a positive experience and we intend to participate again in 2009.

References

[1]. The *Genetic and Evolutionary Computation Conference (GECCO)* 2008, July 12-16, 2008, Atlanta, Georgia. Contest Problems: Finding a Balanced Diet in a Fractal world. URL: http://marvin.cs.uidaho.edu/~heckendo/Gecco08Contest/FractalFoodProblem/

[2]. J. R. Koza (1992): *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* The MIT Press.

[3]. P. Chopra (2008): *Genetic Programming System in Python*. URL: http://www.paraschopra.com/sourcecode/GP/index.php

Appendix 1. Actual GP output translated into the function **walk_move.**

```
+--SUB
| +--[X]
| +--ADD
| | +--DIV
| | | +--COS
| | | | +--PERC
| | | | | +--MUL
| | | | | | +--[X]
| | | | | | +--[20]
| | | | | +--[Y]
| | | +--[11]
| | +--SQRT
| | | +--ADD
| | | | +--ADD
| | | | | +--[15]
| | | | | +--[Y]
| | | | +--PERC
| | | | | +--[X]
| | | | | +--POWER
| | | | | | +--SUB
| | | | | | | +--[Y]
| | | | | | | +--DIV
| | | | | | | | +--ADD
| | | | | | | | | +--[Y]
| | | | | | | | | +--PERC
| | | | | | | | | | +--[X]
| | | | | | | | | | +--[Y]
| | | | | | | | +--MUL
| | | | | | | | | +--COS
| | | | | | | | | | +--DIV
| | | | | | | | | | | +--[21]
| | | | | | | | | | | +--[Y]
| | | | | | | | | +--SUB
| | | | | | | | | | +--[Y]
| | | | | | | | | | +--PERC
| | | | | | | | | | | +--DIV
| | | | | | | | | | | | +--COS
| | | | | | | | | | | | | +--[Y]
| | | | | | | | | | | | +--[8]
| | | | | | | | | | | +--[16]
| | | | | | +--POWER
| | | | | | | +--SQRT
| | | | | | | | +--[8]
| | | | | | | +--[1]
```

Appendix 2. Actual GP output translated into the function **game_walk**

```
+--ADD
| +--POWER
| | +--OR
| | | +--DIV
| | | | +--OR
| | | | | +--AND
| | | | | | +--ADD
| | | | | | | +--SUB
| | | | | | | | +--[13]
| | | | | | | | +--[Y]
| | | | | | | +--ELEV
| | | | | | | | +--MUL
| | | | | | | | | +--[13]
| | | | | | | | | +--[LAST]
| | | | | | +--[24]
| | | | | +--[X]
| | | | +--WALKF
| | | | | +--[22]
| | | | | +--[Y]
| | | +--[GAME]
| | +--[17]
| +--[20]
```

Appendix 3. Actual GP output translated into the function **walk_elev_grain**

```
+--PERC
| +--[21]
| +--ADD
| | +--SUB
| | | +--[Y]
| | | +--PERC
| | | | +--[12]
| | | | +--SUB
| | | | | +--[17]
| | | | | +--WALKF
| | | | | | +--SQRT
| | | | | | | +--[Y]
| | | | | | +--WALKF
| | | | | | | +--SUB
| | | | | | | | +--[17]
| | | | | | | | +--WALKF
| | | | | | | | | +--SQRT
| | | | | | | | | | +--[Y]
| | | | | | | | | +--WALKF
| | | | | | | | | | +--MUL
| | | | | | | | | | | +--[7]
| | | | | | | | | | | +--[5]
| | | | | | | | | | +--[Y]
| | | | | | | +--[Y]
| | +--ELEV
| | | +--[GRAIN]
```

Appendix 4. Actual GP output translated into the function **three_heads**

```
Condition tree:
+--LT
|  +--[19]
|  +--[X]
True tree:
+--SQRT
|  +--WALKF
|  |  +--[18]
|  |  +--SQRT
|  |  |  +--SUB
|  |  |  |  +--[7]
|  |  |  |  +--SUB
|  |  |  |  |  +--DIV
|  |  |  |  |  |  +--[4]
|  |  |  |  |  |  +--[4]
|  |  |  |  |  +--MUL
|  |  |  |  |  |  +--SUB
|  |  |  |  |  |  |  +--PERC
|  |  |  |  |  |  |  |  +--POWER
|  |  |  |  |  |  |  |  |  +--[6]
|  |  |  |  |  |  |  |  |  +--[8]
|  |  |  |  |  |  |  |  +--COS
|  |  |  |  |  |  |  |  |  +--[GAME]
|  |  |  |  |  |  |  +--[19]
|  |  |  |  |  |  +--[24]
False tree:
+--SQRT
|  +--MUL
|  |  +--[Y]
|  |  +--[LAST]
```