EVOLUTION OF FOOD FORAGING STRATEGIES FOR THE CARIBBEAN *ANOLIS* LIZARD USING GENETIC PROGRAMMING

John R. Koza

Computer Science Department

Margaret Jacks Hall

Stanford University

Stanford, California 94305 USA

Koza@Sunburn.Stanford.Edu

415-941-0336

James P. Rice

Stanford University Knowledge Systems Laboratory

701 Welch Road

Palo Alto, California 94304 USA

Rice@Sumex-Aim.Stanford.Edu

415-723-8405

Jonathan Roughgarden

Department of Biological Sciences

Herrin Hall

Stanford University

Stanford, California 94305 USA

Rough@Pangea.Stanford.Edu

415-723-3648

Abstract

This paper describes the recently developed genetic programming paradigm which genetically breeds a population of computer programs to solve problems. The paper then shows, step by step, how to apply genetic programming to a problem of behavioral ecology in biology – specifically, two versions of the problem of finding an optimal food foraging strategy for the Caribbean *Anoli*s lizard. A simulation of the adaptive behavior of the lizard is required to evaluate each possible adaptive control strategy considered for the lizard. The foraging strategy produced by genetic programming is close to the mathematical solution for the one version for which the solution is known and appears to be a reasonable approximation to the solution for the second version of the problem.

Acknowledgments

The collaborative work contained in this paper is the outgrowth of the Founding Workshop in Adaptive Computation held at the Santa Fe Institute on March 10-15, 1992. This paper has been

submitted to the *Simulation of Adaptive Behavior* journal published by The MIT Press.

1 Introduction and Overview

Organisms in nature often possess an optimally designed anatomical trait. For example, a bird's wing may be shaped to maximize lift or a leaf may be shaped to maximize interception of light. In such instances, natural selection caused the population to accumulate genes whose expression led to the observed optimal anatomy. But what of behavior? Certainly in most vertebrates and many invertebrates, behavior is not a direct expression of genes, but instead reflects the result of experience accumulated during the animal's life. Ecologists have, over the years, observed numerous behaviors in nature that closely agree with analytical calculations of optimal behavior or near optimal behavior. The question arises as to whether such optimal behavior be attained by means of accumulating environmental experience. This paper offers a demonstration of one way in which optimal behavior, specifically optimal behavior in foraging for food, may be attained as a result of accumulating environmental experience during the animal's life.

The green, grey, and brown lizards of the genus *Anolis* in the Caribbean islands occupy the ecological niche occupied in North America and Europe by ground-feeding insectivorous birds, such as robins and blue jays. These anoles are "sit and wait" predators typically perch head-down on tree trunks and scan the ground for desirable insects to eat (Ehrlich and Roughgarden 1987, Roughgarden 1989). Figure 1 shows an anole perched head-down on a tree trunk.

Figure 1 Anole lizard perched head-down on a tree trunk.

The optimal foraging strategy for such lizards in their environment is the behavioral rule which, when followed repetitively by a lizard, yields the maximum amount of food for the lizard. A possible foraging strategy is to attack prey according to a criteria or policy that minimizes the average time invested per captured item of prey.

Insects appear probabilistically within the lizard's viewing area. The lizard sees all insects that are in the 180° planar area visible from the lizard's perch. If insects only rarely alight within the lizard's viewing area, it would seem advantageous for the lizard unconditionally to chase every insect that it sees. If insects are abundant, the lizard should certainly chase every nearby insect. However, if insects are abundant and the lizard chases a distant insect, the lizard will be away from its perch for so long that it will forgo the possibility of chasing and eating a greater number of nearby insects. This suggests ignoring distant insects. However, there is no guarantee that any insects will appear nearby during the period of time just after the lizard decides to forgo a distant insect.

The question arises as to what is the optimal tradeoff among these competing considerations. The optimal strategy for the lizard is a function of four variables, namely the probability of appearance of the prey per square meter per second (called the abundance *a*), the lizard's sprint velocity *v* (in meters per second), and the location of the insect within the lizard's planar viewing area (expressed via its two x-y coordinates). Thus, this problem involves finding a function of four variables that optimizes the average time spent per food item eaten by the lizard. Determining what food is eaten when the lizard pursues a particular foraging policy requires a simulation of adaptive behavior.

In section 2, we analytically derive an optimal control strategy for the lizard. In section 3, we provide background on genetic algorithms. In section 4, we describe the recently developed genetic programming paradigm. In section 5, we show the steps required to prepare to use genetic programming to solve a problem. In section 6, we use genetic programming to genetically breed a foraging strategy for a version of this problem for which the optimal strategy is known. In section 7, we breed a foraging strategy for which the optimal strategy is not known.

2 The Optimal Foraging Strategy

Time is used while the lizard waits for prey to appear and while the lizard chases the insect and returns to its perch. We consider a model in which all prey items are identical and the amount of time spent waiting for prey is calorically equivalent to a unit of time spent chasing a prey, so that minimizing time maximizes the energy yield per unit of time. A physiologically more elaborate energy model for the lizard appears in Roughgarden 1992. Moreover, for the values of abundance appropriate to this model, insects are sufficiently rare that the lizard can be assumed to start every chase from its perch.

In the first version of this problem, the lizard always catches the insect if the lizard decides to chase the insect. The functional form of the optimal strategy for the lizard for this version of the problem must be a semicircle. Thus, this version of the problem reduces to finding the cutoff radius r_C for the semicircle such that insects are chased if they are closer than this value and ignored if they are farther than this value.

The average waiting time between the appearance of insects within the semicircle of radius *r* is

$$\frac{1}{\int_0^{r_c} a\pi r dr}.$$

The average pursuit time is the integral from 0 to r_C of the product of the probability that an insect is at distance *r* times the pursuit time, *2r/v*, for the insect at distance *r*, namely

$$\int_0^{r_c} \frac{a\pi r}{\int_0^{r_c} a\pi r\,dr} \frac{2r}{v} dr$$

The average waiting time w spent per insect captured is the sum of the average pursuit time and the average waiting time between the appearance of insects, namely

$$w = \frac{1}{\int_0^{r_c} a\pi r dr} + \int_0^{r_c} \frac{a\pi r}{\int_0^{r_c} a\pi r dr} \frac{2r}{v} dr \; .$$

For example 1 of this problem, Roughgarden was able to do the integration required and obtain

$$\mathrm{w} = \frac{2}{a\pi r_c{}^2} + \frac{4r_c}{3v}$$

The optimal foraging distance r^* is the value of r_C that minimizes w. The minimum value of w occurs when the cutoff radius r_C is equal to

$$\sqrt[3]{(\frac{3v}{\pi a})}.$$

The optimal control strategy for specifying when the lizard should decide to chase an insect can be expressed in terms of a function returning +1 for a point *(x,y)* in the lizard's viewing area for which it is advisable for the lizard to initiate a chase and returning -1 for points for which it is advisable to ignore the insect. Thus, if an insect appears at position *(x,y)* in the 180° area visible from the lizard's perch (0,0), the optimal foraging strategy is

$$\mathrm{Sig}\ [\sqrt[3]{(\frac{3v}{\pi a})} - \sqrt[2]{(x^2 + y^2)}].$$

where Sig is the sign function that returns +1 for a positive argument and –1 otherwise. That is, the lizard should chase the insect if the insect lies inside the semicircle centered at the lizard's perch of radius r^*.

Figure 2 shows this optimal foraging strategy via the switching curve (i.e., semicircle) which partitions the half plane into the +1 (chase) region and the –1 (ignore) region. In this figure, we show an insect at position (x_1,y_1) that is in the –1 (ignore) region of the lizard's 20 meter by 10 meter viewing area.

Figure 3 shows the result of applying the optimal control strategy for one experiment lasting 300 seconds in the particular case where the probability of appearance of the prey (i.e., the abundance a) is 0.003 per square meter per second and where the lizard's sprint velocity v is 1.5 meters per second. Of the 180 insects shown as dots that appear in this 200 square meter area during this 300-second experiment, 91 are inside the semicircle and about 89 are outside the semicircle. Thirty-one of the 91 insects inside the semicircle are actually chased and eaten and are shown as larger dots. Sixty of the 91 insects appear in the semicircular "chase" region while the lizard is away from its perch and are shown as small dots.

Finding the above mathematical expression in closed form for the optimal strategy for this first version of the foraging problem depended on the realization that the functional form of the solution was a semicircle and our being able to perform the required integration.

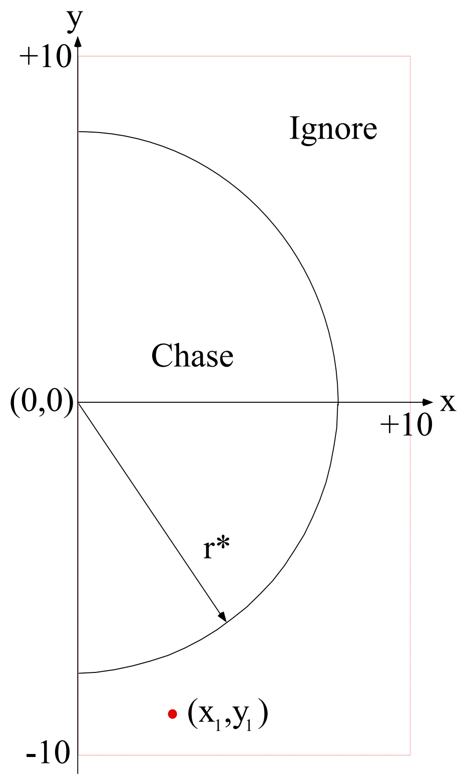

Figure 2 Switching curve for optimal foraging strategy for example 1.

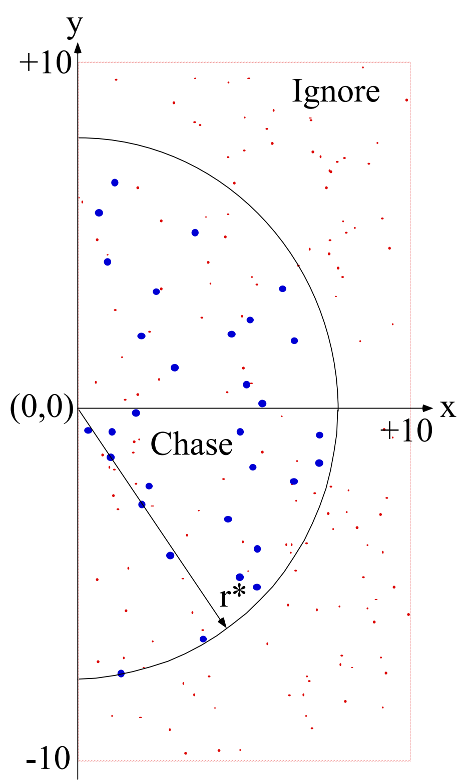

Figure 3 Performance of the optimal foraging strategy for example 1.

3 Background on Genetic Algorithms

John Holland's pioneering *Adaptation in Natural and Artificial Systems* [1975] described how the evolutionary process in nature can be applied to artificial systems using the genetic algorithm operating on fixed length character strings. Holland demonstrated that a population of fixed length character strings (each representing a proposed solution to a problem) can be genetically bred using the Darwinian operation of fitness proportionate reproduction and the genetic operation of recombination. The recombination operation combines parts of two chromosome-like fixed length character strings, each selected on the basis of their fitness, to produce new offspring strings. Holland established, among other things, that the genetic algorithm is a mathematically near optimal approach to adaptation in that it maximizes expected overall average payoff when the adaptive process is viewed as a multi-armed slot machine problem requiring an optimal allocation of future trials given the currently available information.

Genetic algorithms are an efficient way to search a highly non-linear multi-dimensional space. A good overview of the many practical applications of the genetic algorithm operating on fixed length character strings (and other variants of the genetic algorithm) can be found in Goldberg [1989], Davis [1987,1991], Belew and Booker [1991], and Rawlins [1991]. Applications of the genetic algorithm involving simulation of adaptive behavior can be found in Meyer and Wilson [1991].

4 Background on Genetic Programming

For many problems, the most natural representation for solutions are computer programs. The size, shape, and contents of the computer program to solve the problem is generally not known in advance. The computer program that solves a given problem is typically a hierarchical composition of various functions and typically takes the state variables of the system as inputs.

In *Genetic Programming: On the Programming of Computers by Means of Natural Selection and Genetics* [Koza 1992a] and elsewhere [Koza 1989, 1992b], we have shown that computer programs can be genetically bred to solve problems in a surprising variety of areas, including

- optimal control (e.g., centering a cart and balancing a broom on a moving cart in minimal time by applying a "bang bang" force to the cart [Koza and Keane 1990] and backing a tractor-trailer truck to a loading dock [Koza 1992a],
- planning (e.g., navigating an artificial ant along a trail) [Koza 1991a],
- finding minimax strategies for games (e.g., differential pursuer-evader games, discrete games in extensive form) by both evolution and co-evolution [Koza 1991b],
- evolving robotic action plans in the style of the subsumption architecture (e.g., a wall following strategy for a robot with sonar sensors in an irregular room) [Koza 1992d],
- empirical discovery (e.g., rediscovering Kepler's Third Law, rediscovering the well-known non-linear econometric "exchange equation" MV = PQ from actual, noisy time series data for the money supply, the velocity of money, the price level, and the gross national product of an economy),
- discovering inverse kinematic equations (e.g., to move a robot arm to designated target points),
- simultaneous architectural design and training of neural networks [Koza and Rice 1991a].

A videotape visualization of various applications of genetic programming can be found in Koza and Rice [1991b, 1992].

4.1 Objects used in Genetic Programming

In genetic programming, the individuals in the population are compositions of functions and terminals appropriate to the particular problem domain. The set of functions used typically includes arithmetic operations, mathematical functions, conditional logical operations, and domain-specific functions. The set of terminals used typically includes inputs appropriate to the problem domain and various constants. Each function in the function set should be well defined for any combination of elements from the range of every function that it may encounter and every terminal that it may encounter.

The compositions of functions and terminals described above correspond directly to the parse tree that is internally created by most compilers and to the programs found in programming languages such as LISP (where they are called symbolic expressions or S-expressions).

In genetic programming, we view the search for a solution to the problem as a search in the space of all possible compositions of functions that can be recursively composed of the available functions and terminals.

4.2 Steps Required to Execute the Genetic Programming

Genetic programming, like the conventional genetic algorithm, is a domain independent method. It proceeds by genetically breeding populations of computer programs by executing the following three steps:

(1) Generate an initial population of random compositions of the functions and terminals of the problem (i.e., computer programs).
(2) Iteratively perform the following sub-steps until a termination criterion has been satisfied:
  (a) Execute each program in the population and assign it a fitness value according to how well it solves the problem.
  (b) Create a new population of computer programs by applying the following two operations. The operations are applied to computer program(s) in the population chosen with a probability based on fitness.
    (i) *Reproduction*: Copy existing computer programs to the new population.
    (ii) *Crossover*: Create new computer programs by genetically recombining randomly chosen parts of two existing programs.
(3) The single best computer program in the population at the time of termination is designated as the result of the run of genetic programming. This result may be a solution (or approximate solution) to the problem.

4.3 Operations used in Genetic Programming

The basic genetic operations in genetic programming are fitness proportionate reproduction and crossover (recombination).

The Darwinian reproduction operation merely involves copying a computer program from the current population into the new population.

The genetic crossover (sexual recombination) operation operates on two parental computer programs and produces two offspring programs using parts of each parent.

For example, consider the following computer program:

`(OR` **`(NOT D1)`** `(AND D0 D1)).`

This program takes two inputs (`D0` and `D1`) and produces a Boolean-valued output which will either be `T` (True) or `NIL` (false). In the prefix notation used, the function `NOT` is first applied to the terminal `D1` to produce an intermediate result. Then, the function `AND` is applied to the terminals `D0` and `D1` to produce a second intermediate result. Finally, the function `OR` is applied to the two intermediate results to produce the overall result (`T` or `NIL`).

Also, consider a second program:

```
(OR (OR D1 (NOT D0))
    (AND (NOT D0) (NOT D1))).
```

In figure 4, these two programs are depicted as rooted, point-labeled trees with ordered branches. Internal points (i.e., nodes) of the tree correspond to functions (i.e., operations) and external points (i.e. leaves, endpoints) correspond to terminals (i.e., input data). The numbers on the function and terminal points of the tree appear for reference only.

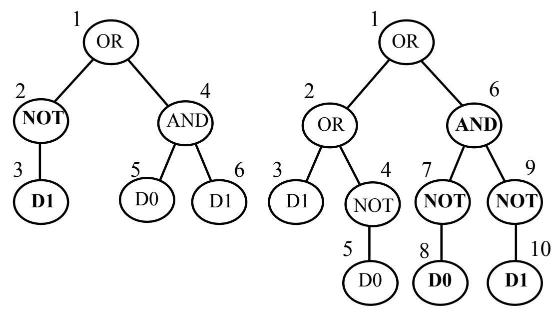

Figure 4 Two Parental computer programs.

The crossover operation creates new offspring by exchanging sub-trees (i.e., sub-lists) between the two parents. Because entire sub-trees are swapped, this crossover operation always produces syntactically and semantically valid programs as offspring regardless of the crossover points.

Assume that the points of both trees are numbered in a depth-first way starting at the left. Suppose that the point no. 2 (out of 6 points of the first parent) is randomly selected as the crossover point for the first parent and that the point no. 6 (out of 10 points of the second parent) is randomly selected as the crossover point of the second parent. The crossover points in the trees above are therefore the `NOT` in the first parent and the `AND` in the second parent. The two crossover fragments are the two sub-trees shown in figure 5.

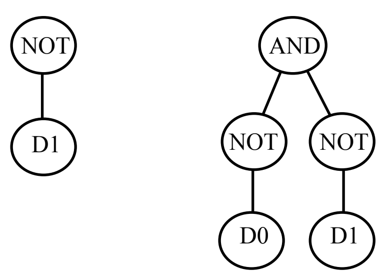

Figure 5 Two Crossover Fragments

These two crossover fragments correspond to the bold, underlined sub-programs (sub-lists) in the two parental computer programs shown in figure 5. The two offspring resulting from crossover are shown in figure 6.

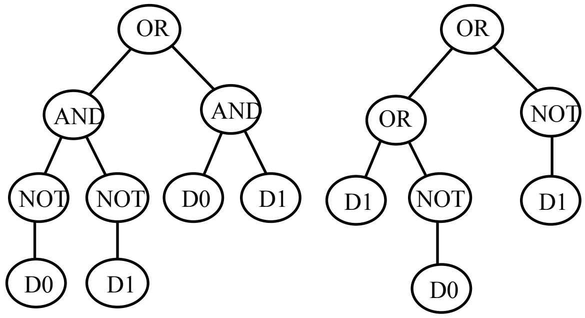

Figure 6 Two Offspring.

Thus, crossover creates new computer programs using parts of existing parental programs. Because programs are selected to participate in the crossover operation with a probability proportional to fitness, crossover allocates future trials to parts of the search space whose programs contains parts from promising programs.

5 Preparatory Steps for Using Genetic Programming

There are five major steps in preparing to use genetic programming, namely determining:

(1) the set of terminals,
(2) the set of functions,
(3) the fitness measure,
(4) the parameters and variables for controlling the run, and
(5) the criterion for designating a result and terminating a run.

We will now apply these five major preparatory steps to the problem of finding an optimal foraging strategy for the lizard.

The computer programs (i.e., control strategies) in the genetic population are compositions of functions and terminals.

The first major step in preparing to use genetic programming is to identify the set of terminals. The four variables (i.e., `X`, `Y`, `AB`, `VEL`) can be viewed as inputs to the unknown computer program which we want to find via genetic programming for optimally controlling the lizard. Here `X` and `Y` represent the position of an insect. `X` and `Y` vary each time an insect appears within a simulation. The value `AB` represents the abundance *a* and `VEL` represents the lizard's sprint velocity *v*. The values of `AB` and `VEL` are constant within any one simulation, but these parameters vary between simulations. Thus, the terminal set T for this problem is

T = {`X`, `Y`, `AB`, `VEL`, ←}.

When the ephemeral random floating point constant ← appears in a program in the initial random population (i.e., generation 0), it is replaced by a different random floating point value between –1.0 and +1.0. Thereafter, that random value remains fixed.

The second major step in preparing to use genetic programming is to identify a sufficient set of functions to solve for the problem. The function set F consisting of four arithmetic operations, the two-argument exponentiation function `SREXPT`, and the decision function `IFLTZ` ("If Less Than or Equal") seems reasonable. That is,

```
F = {+, -, *, %, SREXPT, IFLTE},
```

taking 2, 2, 2, 2, 2, and 4 arguments, respectively.

The protected division function `%` returns one when division by zero is attempted, and, otherwise, returns the normal quotient.

The two-argument exponentiation function `SREXPT` raises the absolute value of the first argument to the power specified by its second argument as a real value. For example, `(SREXPT -2.0 0.5)` returns $\sqrt{2}\,|-2.0| = +1.414$.

Since exponentiation as well as the arithmetic operations have the potential of creating very large or very small floating point values, each function traps any floating point overflow or underflow errors and returns a very large (or very small) fixed arbitrary value.

The conditional branching function `IFLTE` ("If Less Than or Equal") evaluates its third argument if its first argument is less than its second argument and otherwise evaluates its fourth argument. For example, `(IFLTZ 1.5 2.0 P Q)` returns the result of evaluating `P`.

Since `IFLTE` returns a floating point value, `%` protects against division by zero, `SREXPT` cannot produce a complex value, and `SREXPT` and the arithmetic operations cannot produce an overflow or underflow, any composition of these functions will produce a valid result (i.e., there is closure among the functions of the function set).

The third major step in preparing to use genetic programming is the identification of the fitness measure for evaluating how good a given computer program is at solving the problem at hand. In this application, a simulation of the lizard's behavior is required to compute the fitness of a program.

Each program is tested against a simulated environment consisting of 36 combinations of values of the parameters `AB` and `VEL`. The abundance `AB` ranges over six values from 0.0030 to 0.0050 in steps of 0.0004. The lizard's sprint velocity `VEL` ranges over six values from 0.5 meters per second to 1.5 in steps of 0.2. Thirty-six combinations of values of these two parameters are used so as to provide a sufficiently varied environment to permit genetic programming to produce a solution which is likely to generalize to other combinations of values of these two parameters. Creation of the fitness cases for a problem is similar to creating a test set of data for debugging a hand-written computer program.

Because the appearance of insects is probabilistic, the simulation of the lizard's behavior should be done more than once for each of the 36 combinations of values. Computer time was conserved by performing only two experiments for each of the 36 combinations. Thus, there are 72 fitness cases (experiments) for this problem.

A total of 300 seconds of simulated time are provided for each simulation.

The fitness of a program in the population is defined to be the sum, over the 72 experiments, of the number of insects eaten by the lizard. A total of 17,256 insects are available in the 72 experiments. The optimal foraging strategy represented by the closed expression derived above catches about 1,671 insects. This number is only approximate since the insects appear probabilistically in each experiment. Since we do not expect to do better than the known optimal strategy, fitness effectively varies between 0 and about 1,671. The details of converting the above fitness measure into the standardized fitness measure and the normalized fitness measure is discussed in Koza [1992a].

A hit is defined as an experiment for which the number of insects eaten is equal to or greater than one less than the number eaten using the closed-form optimal foraging strategy derived above. That is, a hit indicates that the program has only a small shortfall in performance for a particular experiment with respect to the optimal foraging strategy. Hits range between 0 and 72. Hits are used only to control the termination of a run and for external monitoring of runs. The evolutionary process is driven by fitness, not hits.

Since a given computer program can return any floating-point value and we want our programs to advise the lizard as to whether to chase (+1) or ignore (–1) the current insect, a wrapper (output interface) is used to convert the value returned by a given individual computer program to a value appropriate to this problem domain. In particular, if the program evaluates to any non-negative number, the wrapper returns +1 (chase), but otherwise returns –1 (ignore).

The fourth major step in preparing to use genetic programming is selecting the values of certain parameters. The major parameters are population size and the maximum number of generations to be run. We used a population size of 1,000 individual programs and a maximum number of 61 generations. Our choice of population size and number of generations to be run reflected an estimate on our part as to the likely complexity of the solution to this problem.

In addition, each new generation is created from the preceding generation by applying the fitness proportionate reproduction operation to 10% of the population and by applying the crossover operation to 90% of the population (with both parents selected with a probability proportionate to fitness). In selecting crossover points, 90% were internal (function) points of the tree and 10% were external (terminal) points of the tree. For practical reasons of computer implementation, the depth of initial random programs was limited to 6 and the depth of programs created by crossover was limited to 17. The selection of values of the other parameters are the same as we used on most of the other problems cited in Koza [1992a].

Finally, the fifth major step in preparing to use genetic programming is the selection of the criterion for terminating a run and designating a result. We will terminate a given run when either (i) genetic programming produces a computer program scoring 72 hits, or (ii) 61 generations have been run. We will designate the best program obtained during any generation of the run as the result.

Since great precision was not required by the simulations involved in this problem, we achieved a considerable savings of computer resources on our computer by using the "short float" data type for all numerical calculations.

6 Results for Version 1

Problems of control can be viewed as requiring the discovery of a computer program that takes the variables of a problem as its inputs and produces the value of the control variables as its output. In solving such problems in general, we will usually not be able to identify the functional form of the solution in advance and to do the required integration. When genetic programming is used, there is no need to have any advance insight as to the functional form of the solution and there is no need to do any integration. The solution to a problem produced by genetic programming is not just a numerical solution applicable to a single specific combination of numerical parameters, but, instead, comes in the form of a function (computer program) that maps the variables of the system into values of the control variable. There is no need to specify the exact size and shape of the computer program in advance. The needed structure is evolved in response to the selective pressures of Darwinian natural selection and genetic sexual recombination.

In the first version of this problem, the lizard always catches the insect if the lizard decides to chase the insect.

As one would expect, the performance of the random control strategies found in the initial generation (generation 0) is exceedingly poor. In one run, the worst 4% of the individual computer programs in the population of 1,000 always returned a negative value. Such programs unconditionally advise the lizard not to chase any insects and therefore have a fitness value of zero. An additional 19% of the programs enables the lizard to catch a few insects and scored no hits. 93% of these random programs score two hits or less.

The following individual from a population of 1,000 for generation 0 consisted of 143 points (i.e., functions and terminals) and enables the lizard to catch 1,235 insects:

```
(+ (- (- (* (SREXPT VEL Y) (+ -0.3752 X)) (+ (* VEL 0.991) (+ -0.9522 X))) (IFLTE (+
(% AB Y) (% VEL X)) (+ (+ X 0.3201) (% AB VEL)) (IFLTE (IFLTE X AB X Y) (SREXPT AB
VEL) (+ X -0.9962) (% -0.0542984 AB)) (- (* Y Y) (* Y VEL)))) (% (IFLTE (IFLTE (+ X Y)
(+ X Y) (+ VEL AB) (* Y Y)) (- (% 0.662094 AB) (* VEL X)) (+ (SREXPT AB X) (- X Y))
(IFLTE (* Y Y) (SREXPT VEL VEL) (+ Y VEL) (IFLTE AB AB X VEL))) (IFLTE (IFLTE (SREXPT
X AB) (* VEL -0.0304031) (IFLTE 0.9642 X Y AB) (SREXPT 0.0341034 AB)) (+ (- VEL
0.032898) (- X VEL)) (IFLTE (- X Y) (SREXPT VEL 0.141296) (* X AB) (SREXPT -0.6911
0.5399)) (SREXPT (+ AB AB) (IFLTE 0.90849 VEL AB 0.9308)))))
```

This rather unfit individual from generation 0 is in the 35th percentile of fitness (where the 100th percentile contains the most fit individuals of the population).

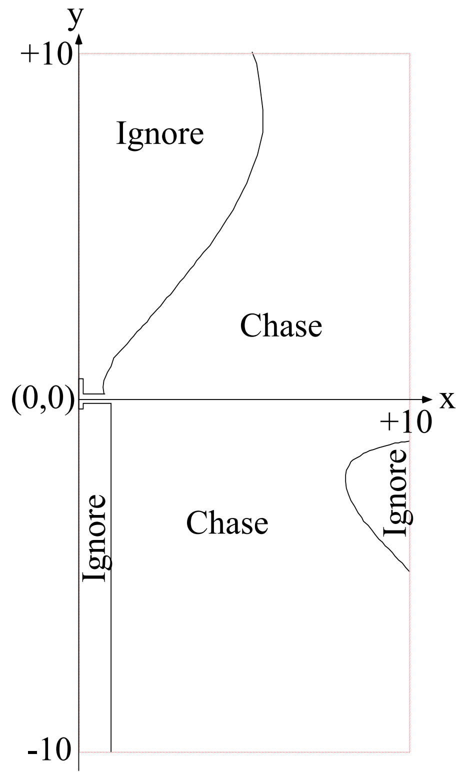

Figure 7 Switching curves of a program from the 35th percentile of fitness for generation 0 for version 1.

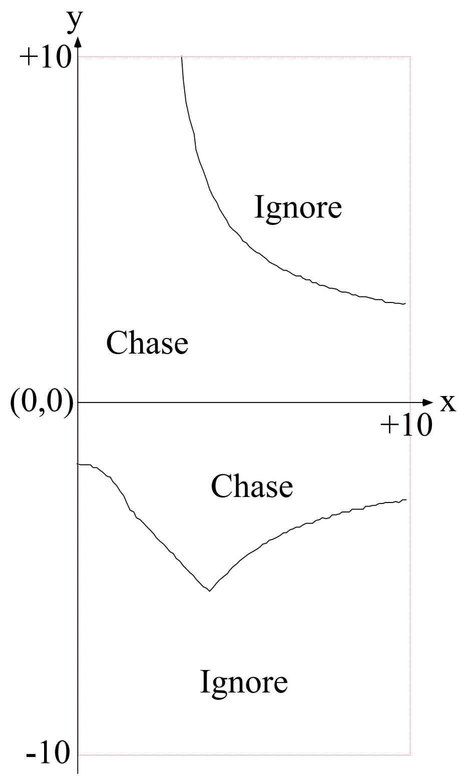

Figure 8 Switching curve of the best-of-generation program from generation 0 for version 1.

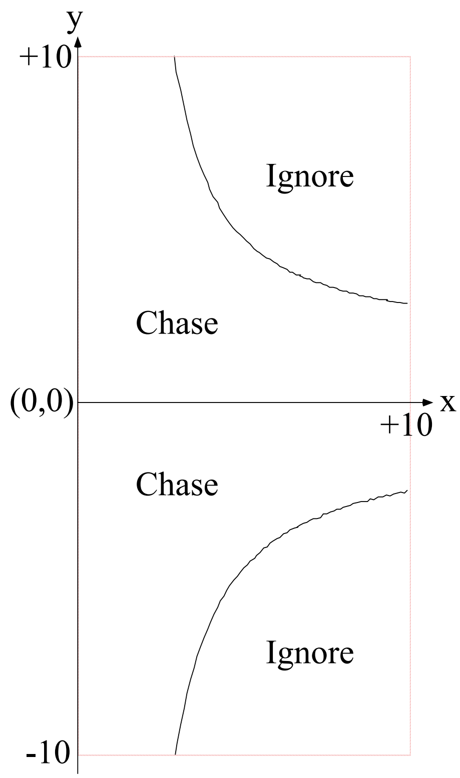

Figure 9 Switching curve of the best-of-generation program from generation 10 for version 1.

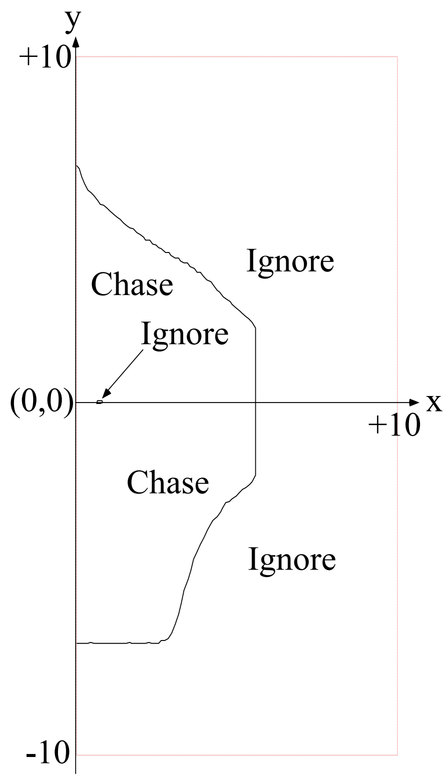

Figure 10 Switching curve of the best-of-generation program from generation 25 for version 1.

Figure 7 graphically depicts the foraging strategy of this individual as a switching curve. This figure and all subsequent figures are based on an abundance `AB` of 0.003 and a sprint velocity `VEL` of 1.5 (i.e., one of the 36 combinations of `AB` and `VEL`). A complete depiction would require showing switching curves for all the other combinations of `AB` and `VEL`. As can be seen, there are three separate "ignore" regions and one large "chase" region. This program causes the lizard to ignore about a third of the insects in the upper half of the figure, including many insects that are very close to the lizard's perch. It also causes the lizard to ignore the thin rectangular region in the lower half of the figure lying along the Y axis. The main part of the "chase" region is distant from the perch, although there is a small T-shaped sliver immediately adjacent to the lizard's perch. Effective foraging behavior involves chasing insects near the perch and ignoring insects that are distant from the perch; this program usually does the opposite.

The best-of-generation individual from generation 0 enables the lizard to catch 1,460 insects. This 37-point program is shown below:

```
(- (- (+ (* 0.5605 Y) (% VEL VEL)) (* (SREXPT Y X) (* X AB))) (* (* (+ X 0.0101929) (*
-0.155502 X)) (IFLTE (+ VEL Y) (- AB X) (* X Y) (SREXPT VEL X))))
```

Figure 8 shows the switching curve for this best-of-generation individual from generation 0. While this non-symmetric control strategy gives poor overall performance, it is somewhat reasonable in that many of the points for which it advises ignoring the insect are distant from the lizard's perch. In particular, all of the points in the "ignore" region at the top of the figure are reasonably distant from the lizard's perch at the origin (0,0) although the boundary is not, by any means, optimal. The "ignore" region at the bottom of the figure gives poorer performance.

However, even in this initial random population, some individuals are better than others. The gradation in performance is used by the evolutionary process to improve the population over subsequent generations. Each successive generation of the population is created by applying the Darwinian operation of fitness proportionate reproduction and the genetic operation of crossover to individuals selected from the population with a probability proportional to fitness.

In generation 10, the best-of-generation individual catches 1,514 insects and scores 26 hits. This 47-point program is shown below:

```
(- (- X (* (SREXPT Y X) (* X AB))) (* (* (+ X 0.0101929) (* -0.155502 (+ AB X)))
(IFLTE (+ X (+ (- (SREXPT X Y) (+ X 0.240997)) (+ 0.105392 VEL))) (% VEL 0.8255) (*
(SREXPT X VEL) (+ -0.7414 VEL)) (SREXPT VEL X))))
```

Figure 9 shows the switching curve for this best-of-generation individual from generation 10. As can be seen, this program advises ignoring the insect when it appears in either of two approximately symmetric regions away from the perch.

In generation 25, the best-of-generation individual catches 1,629 insects and scores 52 hits. This 81-point program is shown below:

```
(- (- (+ (- (- (- (SREXPT AB -0.9738) (SREXPT -0.443604 Y)) (* (SREXPT Y (+ (* (SREXPT
(% (SREXPT Y AB) (- VEL -0.9724)) (+ X 0.0101929)) 0.457596) (+ Y X))) (* X AB))) (*
(* (+ X 0.0101929) (% (+ Y -0.059105) (* 0.9099 Y))) (IFLTE (+ X (SREXPT AB Y)) (% VEL
0.8255) (IFLTE Y VEL 0.282303 -0.272697) (SREXPT (* (SREXPT Y X) (* X AB)) X)))) (% AB
0.412598)) (* (SREXPT X X) (* X AB))) 0.4662)
```

Figure 10 shows the switching curve for this best-of-generation individual from generation 25. In this figure, the control strategy advises the lizard to ignore the insect when the insect is outside an irregular region that vaguely resembles a semicircle centered at the lizard's perch. Note that there is an anomalous close-in point along the *X* axis where this control strategy advises the lizard to ignore any insect.

In generation 40, the best-of-generation individual catches 1,646 insects and scores 60 hits. This 145-point program is shown below:

```
(+ (- (+ (- (- (SREXPT AB -0.9738) (SREXPT -0.443604 Y)) (* (SREXPT X X) (* X AB))) (*
(+ (+ Y -0.059105) (- (SREXPT AB -0.9738) (+ AB X))) (% (- VEL VEL) (+ 0.7457
0.338898)))) (SREXPT Y X)) (- (- (- (SREXPT AB -0.9738) (SREXPT -0.443604 Y)) (*
(SREXPT Y X) (* X AB))) (* (* (+ X 0.0101929) (% (+ (% 0.7717 (+ Y AB)) (SREXPT (IFLTE
Y VEL X Y) (% (+ Y -0.059105) (+ VEL VEL)))) (+ (% (- Y X) (% X AB)) VEL))) (IFLTE (-
(- (- (SREXPT AB -0.9738) (SREXPT -0.443604 Y)) (* (SREXPT X X) (* X AB))) (IFLTE X X
Y AB)) (IFLTE (SREXPT VEL VEL) (+ X 0.0101929) (% VEL -0.407303) (+ -0.496597 AB)) (*
X (SREXPT 0.838104 X)) (SREXPT VEL (+ (+ AB X) (* (% VEL VEL) (IFLTE Y VEL 0.888504
VEL))))))))
```

In generation 60, the best-of-generation individual catches 1,652 insects and scores 62 hits. This 67-point program is shown below:

```
(+ (- (+ (- (SREXPT AB -0.9738) (* (SREXPT X X) (* X AB))) (* (+ VEL AB) (% (- VEL (%
AB Y)) (+ 0.7457 0.338898)))) (SREXPT Y X)) (- (- (SREXPT AB -0.9738) (SREXPT -
0.443604 (- (- (+ (- (SREXPT AB -0.9738) (SREXPT -0.443604 Y)) (+ AB X)) (SREXPT Y X))
(* (* (+ X 0.0101929) AB) X)))) (* (SREXPT Y Y) (* X AB))))
```

This program is equivalent to

$$-0.44^{a + x + a^{-0.9738} - (0.44^{y} + y^{x} + ax[x + 0.01])}$$

$$+ 0.922(v + a)(v - \frac{a}{y})$$

$$+ 2a^{-0.97} \quad -y^{x} \quad -ax(x^{x} + y^{y})$$

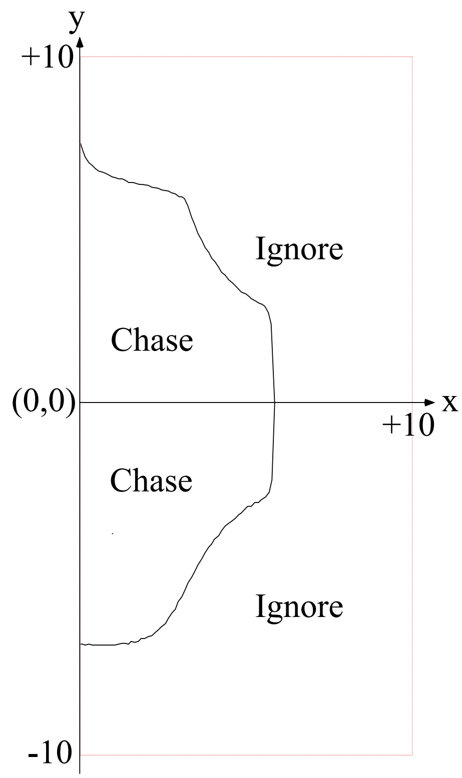

Figure 11 Switching curve of the best-of-generation program from generation 60 for version 1.

Figure 11 shows the switching curve for this best-of-generation individual from generation 60. As before, this figure is based on an abundance AB of 0.003 and a sprint velocity VEL of 1.5. As can be seen, the switching curve here is approximately symmetric and bears a reasonable resemblance to a semicircle centered at the lizard's perch. The score of 1,652 is only 19 short of the 1,671 achieved by the known optimal foraging strategy. In fact, the shortfall from the known optimal strategy is one or less insects for 60 of the 72 fitness cases. Of the remaining 12 fitness cases, eight had a shortfall of only two insects from the known optimal foraging strategy. The performance of this foraging strategy is therefore very close to the performance of the known optimal foraging strategy.

The above control strategy is not the exact solution. It is an approximately correct computer program that emerged from a competitive genetic process that searches the space of possible programs for a satisficing result.

Note also that we did not pre-specify the size and shape of the solution. As we proceeded from generation to generation, the size and shape of the best-of-generation individuals changed as a result of the selective pressure exerted by the fitness measure and the genetic operations. For example, there were 37, 47, 81, 145, and 67 points for the best-of-generation individual for generations 0, 10, 25, 40, and 60, respectively.

The control strategy found in generation 60 of this particular run was obtained after processing 61,000 individuals (i.e., 1,000 individuals for an initial random generation and 60 additional generations). We have achieved similar results in other runs of this problem.

7 Results for Version 2

In the second version of this problem, the lizard does not necessarily catch the insect at the location where it saw the insect.

The lizard's 20 meter by 10 meter viewing area is divided into three regions depending on the probability that the insect will actually be present when the lizard arrives at the location where the lizard saw the insect. Figure 12 shows the three regions. In region I (where the angular location of points lies between -60° and -90°), the insect is never present when the lizard arrives at the location where the lizard saw the insect. In region II (between -60° and the *X* axis), the insect is always present. In region III, the probability that the insect is present when the lizard arrives varies with the angular location of the point within the region. In region III, the probability is 100% along the *X* axis (where the angle is 0°); the probability is 50% along the *Y* axis (where the angle is 90°); and the probability varies linearly as the angle varies between 0° and 90°.

Although we have not attempted to derive a mathematical solution to this version of the problem, it is clear that the lizard should learn to ignore insects it sees in region I and that the lizard should chase insects it sees in region II that are within a circular sector within some radius of the perch. In region III, the lizard should reduce the distance it is willing to travel in pursuit of an insect because of the uncertainty of catching the insect. Moreover, this reduction should depend not just on the insect's distance, but the insect's angular position within region III. This reduction should be greatest for locations on the *Y* axis.

We apply genetic programming in the same manner as in version 1, except that the simulation of the behavior of the lizard must now incorporate the probability of actually catching and eating an insect when the lizard decides to initiate a chase. Since we do not know the optimal foraging strategy for this version of the problem, we do not attempt to define hits. The termination criterion for the run is simply completion of 61 generations.

In one run of this version of the problem, the best-of-generation individual from generation 0 enables the lizard to catch 1,164 insects. This program had 37 points and is shown below:

```
(+ (% (* (IFLTE X VEL VEL X) (+ VEL Y)) (- (% AB X) (+ Y Y))) (SREXPT (* (SREXPT VEL
VEL) (% X Y)) (+ (IFLTE AB VEL 0.194 X) (IFLTE VEL Y VEL VEL))))
```

Figure 13 shows the switching curve of this best-of-generation individual from generation 0. As can be seen, the lizard ignores many locations that are near the *Y* axis. The large gray squares indicate insects which the lizard decides to chase, but which are not present when the lizard arrives. This program is better than the others in generation 0 because it ignores an area in the bottom half of the figure that corresponds roughly to region I and because it ignores the area in the top half of this figure where the probability of catching an observed insect is lowest.

In generation 12, the following 107-point best-of-generation individual enables the lizard to catch 1,228 insects:

```
(IFLTE AB (* (SREXPT (* X 0.71089) (IFLTE VEL AB 0.053299 X)) (+ (- VEL Y) (IFLTE (*
(IFLTE (SREXPT (% -0.175102 (SREXPT Y AB)) (SREXPT (% VEL AB) (IFLTE Y (+ 0.175598 Y)
(+ VEL (+ X (+ AB Y))) (* 0.7769 (IFLTE 0.7204 AB 0.962204 AB))))) VEL (- VEL (SREXPT
(% VEL AB) (% 0.8029 0.36119))) (+ -0.157204 X)) VEL) (- X X) (+ VEL 0.8965) (*
0.180893 AB)))) (+ (IFLTE (- VEL X) (% -0.588 Y) (SREXPT 0.5443 -0.6836) (% X X)) (%
(+ X Y) (- VEL AB))) (- (% (SREXPT AB Y) (IFLTE Y Y X Y)) (IFLTE VEL AB X X)))
```

Figure 14 shows the switching curve of this best-of-generation individual from generation 12. As can be seen, the avoidance of region I and the parts of region III are more pronounced.

In generation 46, the following 227-point best-of-generation individual emerged. It enables the lizard to catch 1,379 insects:

```
(IFLTE AB (* (SREXPT (* -0.588 0.71089) (IFLTE VEL AB 0.053299 X)) (+ (- VEL Y) (IFLTE
(% (SREXPT AB Y) (IFLTE Y Y X Y)) (- VEL (IFLTE X (+ (* AB AB) (+ (IFLTE AB X AB AB)
(* VEL AB))) (* (- X X) AB) AB)) (+ VEL 0.8965) (+ 0.175598 Y)))) (+ (IFLTE (+ VEL
VEL) X (SREXPT 0.5443 -0.6836) (% X X)) (% (+ X Y) (- VEL AB))) (- (* (IFLTE (SREXPT
(SREXPT -0.0914 Y) (IFLTE (+ X Y) (% -0.588 Y) (* AB 0.304092) (% X X))) X (- VEL
(IFLTE Y AB X Y)) (+ -0.157204 (IFLTE (+ (- (% (% X (- VEL (IFLTE Y AB X Y))) (- (%
(SREXPT AB Y) (IFLTE Y Y X Y)) (IFLTE VEL AB X X))) (+ (SREXPT Y Y) (- -0.172798 Y)))
(IFLTE (SREXPT AB X) (- (% AB 0.7782) 0.444794) (* (IFLTE Y (SREXPT (SREXPT X
0.299393) (+ VEL X)) 0.6398 Y) (- X -0.6541)) (IFLTE (- (SREXPT Y 0.4991) (- (%
(SREXPT AB Y) (IFLTE Y Y X Y)) (IFLTE VEL AB X X))) AB (SREXPT VEL (SREXPT (SREXPT X
0.299393) (* -0.3575 X))) (+ (% VEL AB) X)))) VEL (- VEL (- (% (SREXPT AB Y) (+
(SREXPT Y Y) (% VEL VEL))) (IFLTE VEL AB X X))) (+ -0.157204 X)))) VEL) (IFLTE VEL AB
X X)))
```

Figure 15 shows the switching curve of this best-of-generation individual from generation 46. As can be seen, the lizard avoids an area that approximately corresponds to region I; chases insects in region II; and is willing to travel less far to catch an insect in region III. Moreover, the distance the lizard is willing to travel in region III is greatest when the angular location of the insect is near 0° and decreases as the angle approaches 90°.

In an optimization problem where the optimal value of the variable to be optimized is not known in advance and cannot be recognized when it is achieved, judgment must be exercised as to when to stop making additional runs. If several runs appear to plateau at approximately the same value, it may be reasonable to accept the best individual from those runs as the result of applying genetic programming to the problem.

8 Discussion and Conclusions

This paper has shown that genetic programming can be applied to the domain of behavioral ecology in biology. In particular, we have demonstrated the genetic breeding of switching curves for a near optimal strategy for the Caribbean *anolis* lizard for two versions of the food foraging problem.

This demonstration of the applicability of genetic programming to the domain of behavioral ecology in biology has biological significance. Artificial intelligence, including artificial life studies, may provide valuable metaphors for how animals behave adaptively. Specifically, artificial intelligence appears to offer three metaphors to describe how adaptive behavior can come to exist in an animal.

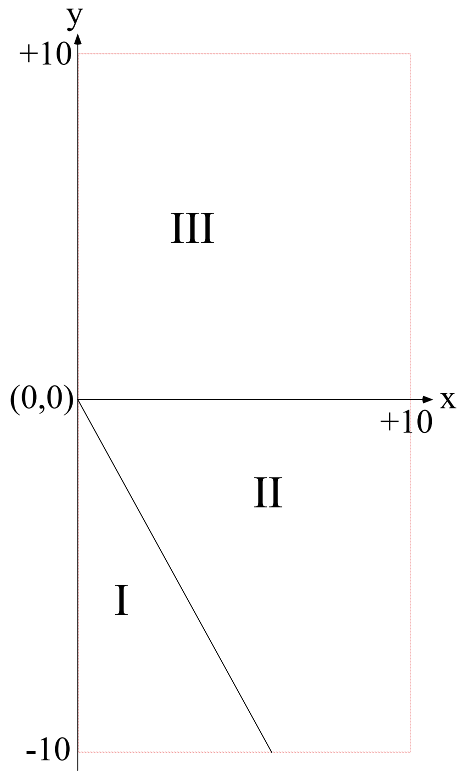

Figure 12 Three regions for version 2.

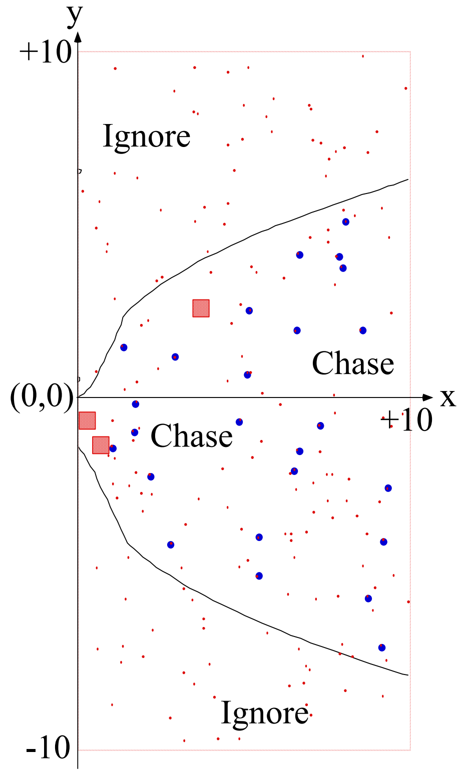

Figure 13 Switching curve of the best-of-generation program from generation 0 for version 2.

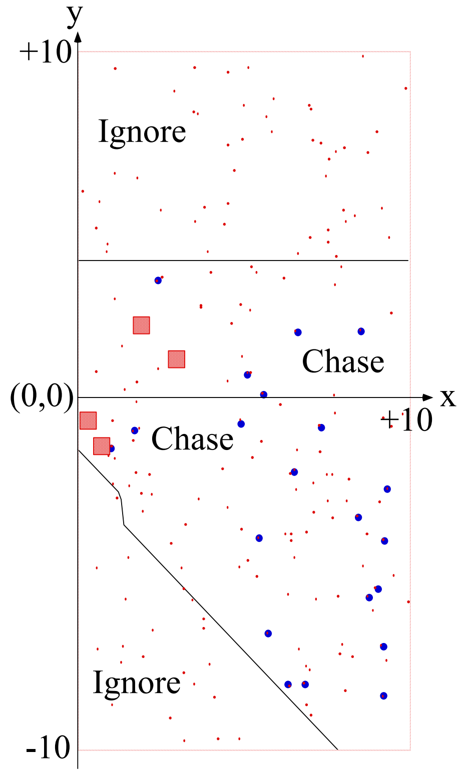

Figure 14 Switching curve of the best-of-generation program from generation 12 for version 2.

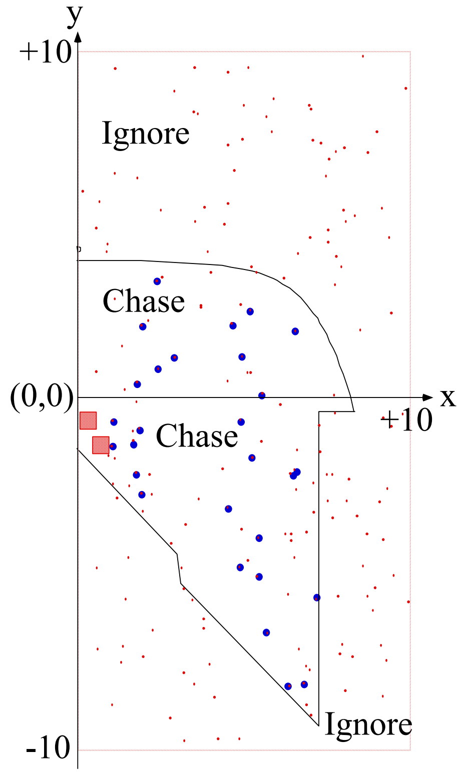

Figure 15 Switching curve of the best-of-generation program from generation 46 for version 2.

First, an animal may instinctually possess a simple decision rule whose repeated application leads to optimal behavior. In Roughgarden 1992, a simple rule is exhibited which, when repeatedly followed, causes an animal's behavior to converge on optimal behavior. This rule was simply conjectured, *a priori*, as being zoologically plausible, with the anticipation of carrying out experiments with lizards in the field to test whether the hypothesized rule is actually employed by lizards.

This paper contributes a second metaphor for how an animal's behavior may approach optimal behavior. Here the rule that the animal follows is not stipulated *a priori*, but is the result of a selection process. This selection can be interpreted in two ways. The population of computer programs on which selection operates may correspond to a population of organisms each possessing a specific behavioral rule. If so, the selection process of genetic programming may be analogous to natural selection acting on a population of biological organisms with respect to a these behavioral rules considered as phenotypic traits. Still, there is a difference. Here the recombination is among the programs themselves. In population genetics, the recombination is among genes, with selection acting on the phenotype that the genes express. But here the programs are the phenotype, so that the recombination here is essentially a recombination among phenotypes. Nonetheless, genetic programming seems analogous to biological evolution, excepting that it is based on a non-traditional genetics.

Alternatively, the development of brain function within a single organism may be thought of as the formation of biochemical or cellular links among various neural modules that represent elementary capabilities. By either this evolutionary or developmental interpretation, the rule that emerges is neither simple nor elegant, but definitely workable. With this metaphor, the function of the rule (i.e., why it was selected) is only remotely related to the properties of any of its component modules examined in isolation. That is, the function of the rule will seem "emergent" because it is effectively unpredictable from inspection of the components of the rule.

A third metaphor for how adaptive behavior can be obtained is supplied by the concept of a neural net. A neural net is a system of inputs and outputs among nodes, and the sensitivity of any one node to another node or to an external input is tunable. When a neural net is trained with a set of test data, the sensitivities of all the interconnections are adjusted to obtain a desired overall system performance. A net might be trained to yield optimal foraging behavior, thus supplying a third metaphor for how an individual organism can attain optimal behavior while experiencing environmental conditions.

These three metaphors for how an animal may come to have optimal behavior seem to represent alternative approaches, not alternative states of nature. That is, an animal may behave optimally by following a simple rule that is itself decomposable into elementary capabilities characteristic of the animal's body plan, and each of these abilities may ultimately be translated into some cellular circuitry. If so, one's preferred metaphor will reflect the purposes of the inquiry, rather than one's beliefs about the state of nature.

9 References

Belew, Richard, and Booker, Lashon (editors). *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann 1991.

Davis, Lawrence (editor). *Genetic Algorithms and Simulated Annealing*. London: Pittman l987.

Davis, Lawrence. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.1991.

Ehrlich, Paul R. and Roughgarden, Jonathan. *The Science of Ecology*. New York: Macmillan 1987.

Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley l989.

Holland, John H. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press 1975. Second edition: Cambridge, MA: The MIT Press 1992.

Koza, John R. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann 1989. Volume I. Pages 768-774.

Koza, John R. Evolution and co-evolution of computer programs to control independent-acting agents. In Meyer, Jean-Arcady, and Wilson, Stewart W. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior.* Paris. September 24-28, 1990. Cambridge, MA: The MIT Press 1991.Pages 366-375. 1991a.

Koza, John R. Genetic evolution and co-evolution of computer programs. In Langton, Christopher, Taylor, Charles, Farmer, J. Doyne, and Rasmussen, Steen (editors). *Artificial Life II, SFI Studies in the Sciences of Complexity*. Volume X. Redwood City, CA: Addison-Wesley 1991. Pages 603-629. 1991b.

Koza, John R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection and Genetics.* Cambridge, MA: The MIT Press 1992. 1992a.

Koza, John R. The genetic programming paradigm: Genetically breeding populations of computer programs to solve problems. In Soucek, Branko and the IRIS Group (editors). *Dynamic, Genetic, and Chaotic Programming*. New York: John Wiley 1992. Pages 203-321. 1992b.

Koza, John R. A genetic approach to finding a controller to back up a tractor-trailer truck. In *Proceedings of the 1992 American Control Conference*. Evanston, IL: American Automatic Control Council 1992. Pages ---. 1992c.

Koza, John R. Evolution of subsumption using genetic programming. In Varela, Francisco J., and Bourgine, Paul (editors). *Toward a Practice of Autonomous Systems: Proceedings of the first European Conference on Artificial Life*. Cambridge, MA: The MIT Press 1992. Pages ---- 1992d.

Koza, John R., and Keane, Martin A. Genetic breeding of non-linear optimal control strategies for broom balancing. In *Proceedings of the Ninth International Conference on Analysis and Optimization of Systems*. *Antibes, France, June, 1990.* Pages 47-56. Berlin:

Springer-Verlag, 1990. 1990.

Koza, John R., and Rice, James P. Genetic generation of both the weights and architecture for a neural network. In *Proceedings of International Joint Conference on Neural Networks, Seattle, July 1991*. Los Alamitos, CA: IEEE Press 1991. Volume II, Pages 397-404. 1991a.

Koza, John R., and Rice, James P. A genetic approach to artificial intelligence. In Langton, C. G. (editor). *Artificial Life II Video Proceedings*. Addison-Wesley 1991. 1991b.

Koza, John R., and Rice, James P. *Genetic Programming: The Movie*. Cambridge, MA: The MIT Press 1992.

Meyer, Jean-Arcady, and Wilson, Stewart W. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior.* Paris. September 24-28, 1990. Cambridge, MA: The MIT Press 1991.

Rawlins, Gregory (editor). *Foundations of Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann 1991.

Roughgarden, Jonathan. *Theory of Population Genetics and Evolutionary Ecology: An Introduction*. New York: Macmillan Publishing Co., Inc. 1979.

Roughgarden, Jonathan. In Roughgarden, Jonathan, May, Robert M., and Levin, Simon A. *Perspectives in Ecological Theory*. Princeton, NJ: Princeton University Press 1989. Pages 203-226.

Roughgarden, Jonathan. *Anolis Lizards of the Caribbean: Ecology, Evolution, and Plate Tectonics*. Oxford University Press 1992. (In Press).