A GENETIC APPROACH TO THE TRUCK BACKER UPPER PROBLEM AND THE INTER-TWINED SPIRAL PROBLEM

John R. Koza
Computer Science Department
Stanford University
Stanford, CA 94305 USA
Koza@Sunburn.Stanford.Edu 415-941-0336

ABSTRACT: *Neural networks are a biologically motivated problem-solving paradigm that has proven successful in robustly solving a variety of problems. This paper describes another biologically motivated paradigm, namely genetic programming, which can also solve a variety of problems. This paper explains genetic programming and applies it to two well-known benchmark problems from the field of neural networks. The truck backer upper problem is a multi-dimensional control problem and the inter-twined spirals problem is a challenging classification problem.*

1 INTRODUCTION AND OVERVIEW

Anyone who has tried to back up a tractor-trailer truck to a loading dock knows that it presents a difficult problem of control. Nguyen and Widrow (1990) successfully illustrated the capabilities of neural networks by finding a controller for this multi-dimensional control problem. Problems of control can be viewed as requiring the discovery of a computer program (i.e. controller, control strategy) that takes the state variables of a problem as its inputs and produces the values of the control variable(s) as its outputs.

The recently developed genetic programming paradigm is well suited to difficult control problems where no exact solution is known and where an exact solution is not required. When genetic programming solves a problem, it produces a computer program that takes the state variables of the system as input and produces the actions required to solve the problem as output. The solution to a problem produced by the genetic programming paradigm is not just a numerical solution applicable to a single specific numerical combination of states, but, instead, comes in the form of a general function (computer program) that maps the state variables of the system into values of the control variable(s). 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.

Lang and Whitbrock (1989) used a neural network to solve the challenging problem of distinguishing two intertwined spirals. Learning patterns that discriminate among problem solving choices is one approach to problem solving in artificial intelligence. Learning problems often present themselves as problems of classification. In classification problems, the goal is to discern a pattern and to develop a procedure capable of successfully performing the classification. The procedure is typically developed using a sample of data and is considered successful if it learns to correctly classify both the original sample and previously unseen points that are a reasonable generalization of the original sample points.

In this paper, we apply genetic programming to these two well-known benchmark problems from the field of neural networks.

2 BACKGROUND ON GENETIC ALGORITHMS

John Holland's pioneering 1975 *Adaptation in Natural and Artificial Systems* described how the evolutionary process in nature can be applied to artificial systems using the genetic algorithm operating on fixed length character strings (Holland 1975). 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 currently available information. A good overview of current work in the field of genetic algorithms can be found in Goldberg (1989), Davis (1987, 1990), Belew and Booker (1991), and Rawlins (1991).

3 BACKGROUND ON THE GENETIC PROGRAMMING PARADIGM

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 primitive functions and terminals. It typically takes the state variables of the system or independent variables of the problem as its input and produces an external action or value of a dependent variable as its output. It is unnatural and difficult to represent program hierarchies of dynamically varying size and shape with the fixed length character strings generally used in the conventional genetic algorithm.

In the genetic programming paradigm, 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 (sensors) appropriate to the problem domain and possibly various constants. Each function in the function set should be well defined for every combination of values of terminals that it may encounter and every combination of values returned by any function that it may encounter.

One can view the search for the solution to many problems as a search of the hyperspace of all possible compositions of functions and terminals (i.e. computer programs) that can be recursively composed from the available functions and terminals.

The symbolic expressions (S-expressions) of the LISP programming language are an especially convenient way to create and manipulate the compositions of functions and terminals described above. These S-expressions in LISP correspond directly to the parse tree that is internally created by most compilers.

The genetic programming paradigm genetically breeds computer programs to solve problems by executing the following three steps:

(1) Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
(2) Iteratively perform the following sub-steps until the 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 primary 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 two new computer programs by genetically recombining randomly chosen parts of two existing programs.
(3) The best computer program ever attained is designated as the result of the genetic programming paradigm. This result may be a solution (or approximate solution) to the problem.

The basic genetic operations for the genetic programming paradigm are reproduction (e.g. fitness proportionate reproduction) and crossover (recombination). The crossover operation is a sexual operation that operates on two parental LISP S-expressions and produces two offspring S-expressions using parts of each parent. The crossover operation creates new offspring S-expressions 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 LISP S-expressions as offspring regardless of the crossover points.

For example, consider the two parental S-expressions:

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

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

Figure 1 graphically depicts these two S-expressions as rooted, point-labeled trees with ordered branches. The numbers on the points of the tree are for reference only.

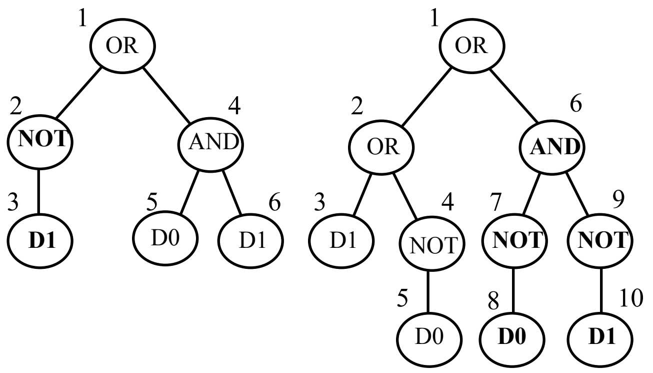


Figure 1 Two parental computer programs shown as trees with ordered branches. Internal points of the tree correspond to functions (i.e. operations) and external points correspond to terminals (i.e. input data).

Assume that the points of both trees are numbered in a depth-first way starting at the left. Suppose that point no. 2 (out of 6 points of the first parent) is randomly selected as the crossover point for the first parent and that 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.

Figure 2 shows the two crossover fragments are two sub-trees. These two crossover fragments correspond to the bold sub-expressions (sub-lists) in the two parental LISP S-expressions shown above. Figure 3 shows the two offspring resulting from the crossover.

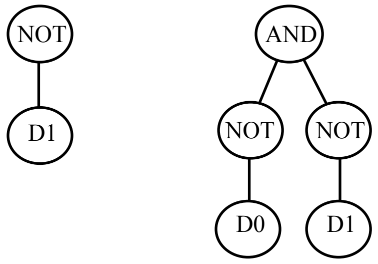


Figure 2 The two crossover fragments.

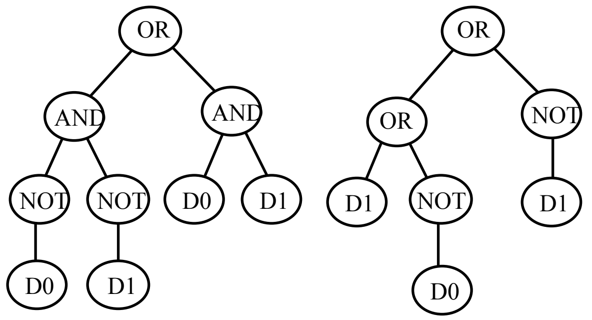


Figure 3 Offspring resulting from crossover.

Note that the first offspring in Figure 2 is an S-expression for the Boolean even-parity (i.e. equal) function, namely

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

Although one might think that computer programs are so epistatic that they could only be genetically bred in a few especially congenial problem domains, we have shown that computer programs can be genetically bred to solve a surprising variety of problems in many different areas [Koza 1989, 1990, 1992a], including

- planning (e.g. navigating an artificial ant along a trail and developing a robotic plan for stacking blocks in to a desired order) [Koza 1989, 1991a],
- emergent behavior (e.g. discovering a computer program which, when executed by all the ants in an ant colony, enables the ants to locate food, pick it up, carry it to the nest, and drop pheromones along the way so as to produce cooperative emergent behavior) [Koza 1991],
- evolution of subsumption (e.g., evolving a program for a wall following robot) [Koza 1992b],
- machine learning of functions (e.g. learning the Boolean 11-multiplexer function) [Koza 1991d],
- automatic programming (e.g. solving pairs of linear equations, solving quadratic equations for complex roots, solving differential equations, and discovering trigonometric identities) [Koza 1992a],
- discovering inverse kinematic equations (e.g. to move a robot arm to designated target points) [Koza 1992a],
- generation of maximal entropy random numbers [Koza 1991e],
- induction of decision trees for classification [Koza 1991b],
- 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 1990a, 1990b],
- symbolic regression, integration, differentiation, and symbolic solution to general functional equations for a solution in the form of a function (including differential equations with initial conditions, and integral equations),
- 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),
- finding minimax strategies for games (e.g. differential pursuer-evader games, discrete games in extensive form) by both evolution and co-evolution, and
- simultaneous architectural design and training of neural networks [Koza and Rice 1991a].

A videotape visualization of the application of the genetic programming paradigm to planning, emergent behavior, empirical discovery, inverse kinematics, and game playing can be found in the *Artificial Life II Video Proceedings* [Koza and Rice 1991b].

There are five major steps in preparing to use the genetic programming paradigm, namely determining (1) the set of terminals, (2) the set of functions, (3) the fitness function, (4) the parameters and variables for controlling the run, and (5) the criterion for designating a result and terminating a run.

4 THE TRUCK BACKER UPPER PROBLEM

The truck backer-upper problem is a four dimensional control problem. Figure 4 shows a loading dock and tractor-trailer. The loading dock is the Y-axis. The trailer and tractor are connected at a pivot point.

The state space of the system is four dimensional. X is the horizontal position of the midpoint of the rear of the trailer and Y is the vertical position of the midpoint. The target point for the midpoint of the rear of the trailer is (0,0). The angle θt (also called TANG) is the angle of the trailer with respect to the loading dock (measured, in radians, from the positive X-axis with counterclockwise being positive). The difference angle θ_d (also called DIFF) is the angle of the tractor relative to the longitudinal axis of the trailer (measured, in radians, from the longitudinal axis of the trailer with counterclockwise being positive).

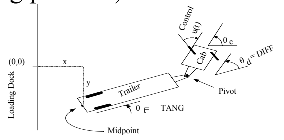


Figure 4: In the truck backer-upper problem, the goal is to bring the midpoint of the rear of the trailer to the target point (0,0) on the loading dock. The control variable is the steering angle u(t) for the tires of the tractor (cab). The cab is connected to the trailer via the pivot.

The truck backs up at a constant speed so that the front wheels of the tractor (cab) move a fixed distance backwards with each time step. Steering is accomplished by changing the angle u (i.e. the control variable) of the front tires of the tractor (cab) with respect to the current orientation of the tractor.

The goal is to guide the midpoint of the rear of the trailer so that it ends up at (or very close to) the target point (0,0) on the loading dock while never allowing the midpoint of the rear of the truck to touch the loading dock. We want to find a control strategy which specifies the change in angle u of the front tires of the tractor (cab) in terms of the four state variables of the system (namely, X, Y, TANG, and DIFF)

The equations of motion that govern the tractor-trailer system are

$$\begin{aligned}
A &= \mathrm{r}\cos u[\mathrm{t}] \\
B &= A\cos(\theta_{\mathrm{c}}[t] - \theta t[t]) \\
C &= A\sin(\theta_{C}[\mathrm{t}] - \theta t[\mathrm{t}]) \\
\mathrm{x}[t+1] &= \mathrm{x}[t] - B\cos\theta t \\
\mathrm{y}[t+1] &= \mathrm{y}[t] - B\sin\theta t \\
\theta_{\mathrm{c}}[t+1] &= \tan^{-1}\ \textbf{Error!}) \\
\theta t[t+1] &= \tan^{-1}\ \textbf{Error!}) \\
\theta_d[t] &= \theta_t[t] - \theta_c[t]
\end{aligned}$$

In these equations, $\tan^{-1}\left(\frac{\mathrm{x}}{\mathrm{y}}\right)$ is the two argument arctangent function (also called ATG here) delivering an angle in the range $-\pi$ to π. The length of the tractor (i.e. cab) d_{c} is 6 meters and the length of the trailer d_{s} is 14 meters. As in Nguyen and Widrow (1990), the truck only moves backwards. The distance moved in one time step is r. The angle θ_t is TANG. The angle of the tractor relative to the X axis is θ_{c}.

The first major step in preparing to use the genetic programming paradigm is to identify the set of terminals. The four state variables of the system (i.e. X, Y, TANG, DIFF) can be viewed as inputs to the unknown computer program which we want to find for controlling the system. Thus, the terminal set T for this problem is T = {X, Y, TANG, DIFF, ←}. When the initial population of random individuals is created, every occurrence of this ephemeral random constant in an S-expression is replaced by a separately generated random floating point number in the range between -1.000 and +1.000.

The second major step in preparing to use the genetic programming paradigm is to identify a sufficient set of functions to solve for the problem. We do not know the solution to this problem. We have no assurance that a chosen function set will be sufficient for the problem. However, the function set F consisting of four arithmetic operations, the two argument Arctangent function ATG, and the conditional comparative operator IFLTZ ("If Less than Zero") seems reasonable. Thus, the function set for this problem is F = {+, -, *, %, ATG, IFLTZ}. taking 2, 2, 2, 2, 2, and 3 arguments, respectively. The protected division function % returns one when division by zero is attempted, and, otherwise, returns the normal quotient. The conditional branching function IFLTZ ("If Less than Zero") evaluates its third argument is its first argument is less than zero and otherwise evaluates its second argument. Since IFLTZ returns a floating point value and % protects against division by zero, there is closure among the functions of the function set.

In selecting this function set, we included the Arctangent function ATG because we thought it might be useful in computing angles from the various distances involved in this problem and we included the conditional comparative operator IFLTZ so that actions could be made conditional on certain conditions being satisfied. As it developed, the Arctangent function did not appear in the best solution we found. It is, of course, necessary to choose a function set and terminal set that are capable of solving the problem at hand. We had no assurance, in advance, that this function set and terminal set would sufficient for a solution.

The third major step in preparing to use the genetic programming paradigm is the identification of the fitness measure for evaluating how good a given computer program is at solving the problem at hand. For this problem, fitness is an error measure. Each program is tested against a simulated environment consisting of eight fitness cases, each consisting of a set of initial conditions for X, Y, and TANG. X is either 20 or 40 meters. Y is either -50 or 50 meters. TANG is either - $\pi/2$ or + $\pi/2$. As in Nguyen and Widrow (1990), the difference angle DIFF is initially always zero (i.e. the tractor and trailer are initially coaxial).

Time is measured in time steps of 0.02 seconds. A total of 3000 time steps (i.e. 60 seconds) are allowed for each fitness case The speed of the tractor-trailer is 0.2 meters per time step. Termination of a fitness case occurs when (1) time runs out, (2) the trailer crashes into the loading dock (i.e. X becomes zero), or (3) the midpoint of the rear of the trailer comes close to the target (0,0) point. A hit for this problem occurs when the value of X is less than 0.1 meters, the absolute value of Y is less than 0.42 meters, and the absolute value of TANG is less than 0.12 radians (i.e. about 14 degrees).

Fitness is the sum, over the fitness cases, of the sum of the squares of the differences, at the time of termination of the fitness case, between the value of X and the target value of X (i.e. 0), the difference between the value of Y and the target value of Y (i.e. 0), and difference between the value of TANG and the target value of TANG (i.e. 0).

A wrapper (output interface) is used to convert the value returned by a given individual computer program to a value appropriate to the problem domain. In particular, if the program evaluates to a number between -1.0 and +1.0, the tractor turns its wheels to that particular angle (in radians) relative to the longitudinal axis of the tractor and backs up for one time step at a constant speed. Outside that range the control variable saturates.

As in Nguyen and Widrow (1990), if a choice of the control variable u would cause the absolute value of difference DIFF to exceed 90 degrees, DIFF is constrained to 90 degrees to prevent jack-knifing.

The fourth major step in preparing to use the genetic programming paradigm is selecting the values of certain parameters. The population size is 1,000 here. 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 the practical reason of conserving computer time, the depth of initial random programs was limited to 4 and the depth of programs created by crossover was limited to 15. Our choice of population size reflected an estimate on our part as to likely complexity of the solution to this problem. The individuals in the initial random generation were generated so as to obtain a wide variety of different sizes and shapes among the S-expressions. The "elitist strategy "was not used. Fitness is "adjusted" to emphasize small differences near zero. Spousal selection was also fitness proportionate. Details can be found in Koza (1992). The author believes that sufficient information is provided herein (and in the references cited herein) to allow replication of the experimental results reported herein, within the limits inherent in a probabilistic algorithm.

Finally, the fifth major step in preparing to use the genetic programming paradigm is the selection of the criterion for terminating a run and accepting a result. We will terminate a given run when either (i) the genetic programming paradigm produces a computer program for which all eight fitness cases terminate according to condition (3) above, or (ii) 51 generations have been run.

In one run, the best single individual computer program in the initial population of 1,000 randomly created individual programs was, as one would expect, incapable of backing the tractor-trailer to the loading dock for any of the eight initial conditions (fitness cases) of the tractor-trailer truck. This best-of-generation individual program had an enormous value of fitness, namely 26,956. This S-expression has 19 points and is shown below:

```
(- (ATG (+ X Y) (ATG X Y)) (IFLTZ (- TANG X) (IFLTZ Y TANG TANG) (* 0.3905
DIFF)))
```

However, even in generation 0, some individuals are better than others. And, in the valley of the blind, the one-eyed man is king. The Darwinian operation of fitness proportionate reproduction and genetic crossover (sexual recombination) is now applied to the population, and a new generation of 1,000 S-expressions is produced.

In the next few generations, fitness began to improve (i.e. drop) substantially. It dropped to 4790 for generations 1 and 2, 3131 in generation 3, and 228 for generations 4 and 5. Moreover, in addition to coming closer to the loading dock, for generations 4 and 5, the best-of-generation individual was successful in backing up the truck for one of the eight fitness cases.

Fitness improved to 202 for generation 6. By generation 11, fitness had improved to 38.9 and the best-of-generation individual was successful for three of the eight fitness cases. Between generations 14 and 21, fitness for the best-of-generation individual ranged between 9.99 and 9.08 and the best-of-generation individual was successful for five fitness cases. Between generation 22 and 25, fitness for the best-of-generation individual ranged between 8.52 and 8.47 and the best-of-generation individual was successful for seven fitness cases. Of course, the vast majority of individual computer programs in the population of 1000 were still ineffective in solving the problem (although average performance is also improving).

In generation 26, the fitness of the best-of-generation individual had improved to 7.41. This best-of-generation control strategy was capable of backing up the tractor-trailer to the loading dock for all eight fitness cases. This computer program has 108 points (i.e. functions and terminals) and is shown below.

```
(% (+ (+ (IFLTZ Y Y (+ (% (+ (+ (+ (+ (+ (IFLTZ DIFF Y (% Y TANG)) (- DIFF
X)) (+ (- -0.0728 Y) (% Y TANG))) (- DIFF X)) (+ (- -0.0728 Y) (IFLTZ DIFF
Y (% Y TANG)))) (% Y TANG)) TANG) (- (% (% (+ (+ (IFLTZ Y Y (% Y TANG)) (-
TANG X)) (+ (- -0.0728 Y) (% Y TANG))) TANG) TANG) X))) (- DIFF X)) (+ (+
(+ (+ (+ (IFLTZ DIFF Y (% Y TANG)) (- DIFF X)) (+ (- -0.0728 Y) (% Y
TANG))) (- DIFF X)) (+ (- -0.0728 Y) (% Y TANG))) (% Y TANG))) TANG)
```

No mathematically exact solution to this problem is known. The above control strategy is almost certainly not the exact solution. However, this genetically created control strategy works. 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.

Interestingly, on 89.6% of the time steps involved in evaluating the above best-of-generation individual from generation 26, the absolute value of the control variable returned by this individual exceeded one. That is, a bang bang change in angle was applied.

Note also that we did not pre-specify the size and shape of the solution. We did not specify that the solution would have 108 points. 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 only 19 points for the best-of-generation individual for generation 0 (i.e. the initial random generation).

Note that the 108 point computer program from generation 26 could easily be encoded into a controller using ones preferred programming language.

On this particular run, we obtained a control strategy satisfying the termination criterion of the problem after processing 27,000 individuals (i.e. 1000 individuals for an initial random generation and 26 additional generations). We have achieved similar results in other runs of this problem.

See also Koza and Keane [1990a, 1990b].

5 THE INTER-TWINED SPIRALS PROBLEM

Lang and Whitbrock (1989) used a neural network to solve the problem of distinguishing two intertwined spirals. In their statement of the problem, the two spirals coil around the origin three times in the x-y-plane. The x-y-coordinates of 97 points from each spiral are given. The problem involves learning to classify each point as to which spiral it belongs.

Figures 4 and 5 show the two spirals. The 97 points of the first spiral are indicated by large or small squares and the 97 points of the second spiral are indicated by large or small circles. The first spiral belongs to class +1 and the second spiral belongs to class -1. The task as defined by Lang and Whitbrock is limited to the 194 points in the three turns of these two spirals and does not involve dealing with points that would lie on a fourth or later turns of the extensions of the same spirals.

The terminal set for this problem consists of the x and y position of the 194 given points. In addition, since we may need numerical constants in a computer program capable of processing the 194 given points, we include the ephemeral random floating point constant $\leftarrow$ in the terminal set. Thus, the terminal set is T = {X, Y, $\leftarrow$}.

As to the function set for this problem, it seems reasonable to try to write a computer program for determining to which spiral a given point belongs in terms of the four arithmetic operations, a conditional comparative function for decision making, and the trigonometric sine and cosine functions. Thus, the function set for this problem is F = (+, -, *, %, IFLTE, SIN, COS}, taking 2, 2, 2, 2, 4, 1, and 1 arguments, respectively.

IFLTE (If-Less-Than-or-Equal) is a four-argument conditional comparative operator that executes its third argument if its first argument is less than its second argument and, otherwise, executes the fourth (else) argument.

Since the S-expressions in the population are compositions of functions and terminals operating on floating point numbers and since the S-expressions in this problem must produce a binary output (+1 or -1) to designate the class, a wrapper (output interface) is required. This wrapper maps any positive value to class +1 and maps any other value to class -1.

The fitness of an individual S-expression is computed using fitness cases. The fitness cases are the 194 x-y coordinates of the given points belonging to the spirals and the class (+1 or -1) associated with each point. Raw fitness (hits) is the number of points (0 to 194) that are correctly classified.

The population size is 10,000 here. We will terminate a given run when either (i) the genetic programming paradigm produces a computer program which scores a raw fitness of 194, or (ii) 51 generations have been run. The best-so-far individual obtained in any generation will be designated as the result of the run.

As one would expect, the individual S-expressions in the initial population of randomly created computer programs are highly unfit in solving the problem. In one run, approximately 31% of the initial random individuals in generation 0 correctly classified precisely 50% of the points (i.e., 97 out of a possible 194 points). Some of these individuals, such as (* (* X X) 0.502), scored 50% by virtue of always returning a value with the same sign and therefore classifying all the points as belonging to one spiral. Others, such as (* X Y) scored 50% by virtue of dividing the space into parts which contain exactly half of the points. In addition, about 30% of the population scored between 88 and 96 hits while about 32% scored between 98 and 106 hits. The worst-of-generation individual from generation 0 scored 71 hits while the best-of-generation individual scored 128 hits.

The best-of-generation individual from generation 0 scored 128 hits out of a possible 194 hits and is below:

```
(SIN (% Y 0.30400002))
```

In generation 1, a partially blind best-of-generation individual works by classifying points into vertical bands of varying width. Because it is particularly effective near the X-axis, it does better than the best-of-generation individual from generation 0.

For generation 3, the best-of-generation individual contained 48 points, scored 139 hits, and incorporated both X and Y. It is shown below:

```
(SIN (- (+ (IFLTE (* X -0.25699997) (* X X) (COS Y) (+ (SIN (COS X)) (+ (*
0.18500006 -0.33599997) (IFLTE Y 0.42000008 X -0.23399997)))) (SIN (SIN
Y))) (+ (IFLTE (% (COS X) (SIN -0.594)) (+ -0.553 Y) (% Y -0.30499995) (+ Y
 X)) (COS (% X 0.5230001)))))
```

Figure 4 shows that the best-of-generation individual from generation 3 does especially well near the origin. The points of one spiral are indicated with boxes and the points of the other spiral are indicated with circles. Correctly classified points are indicated by large boxes or circles.

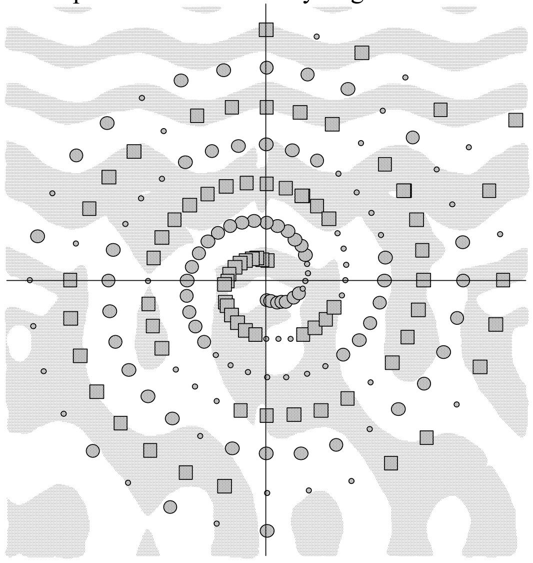


Figure 4 Classification performed by best-of-generation individual from generation 3.

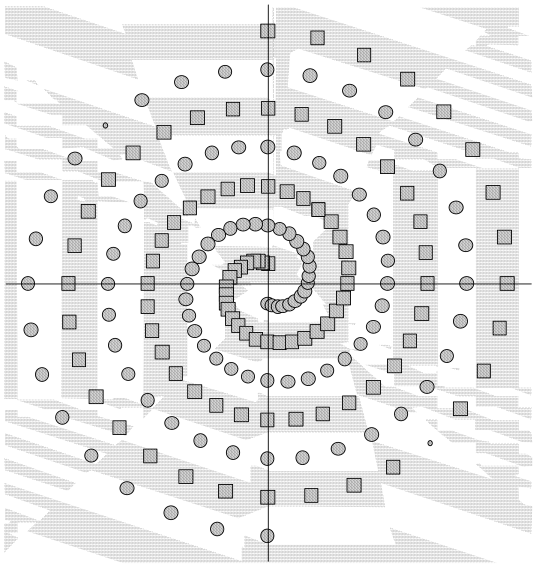


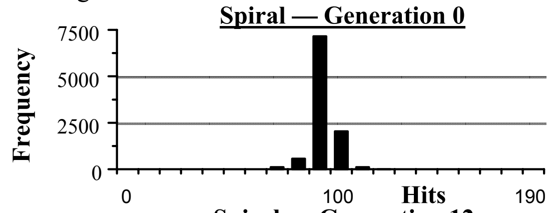
Figure 5 Classification performed by best-of-generation individual from generation 33.

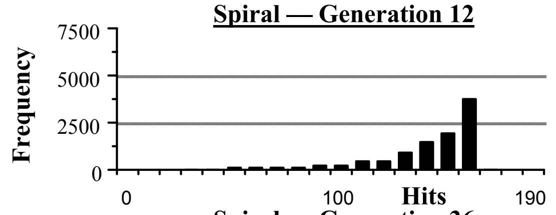
On generation 33, the best-of-generation individual scored 192 out of 194. It had 169 points. Figure 8 shows the classification performed by the best-of-generation individual scoring 192 from generation 33. There are now only two incorrectly classified points in this figure. One is shown as a small circle in a region in the upper left section of the figure which the S-expression incorrectly classified as gray, instead of white. The second is shown as a small square in the lower right section of the figure which the S-expression incorrectly classified as white, instead of gray.

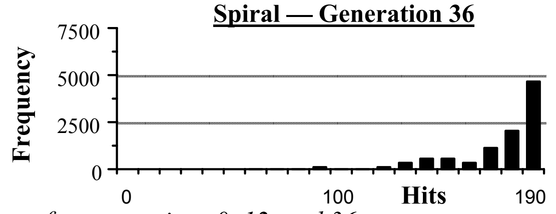
On generation 36, the following S-expression containing 179 points and scoring 194 out of 194 hits emerged:

```
(SIN (IFLTE (IFLTE (+ Y Y) (+ X Y) (- X Y) (+ Y Y)) (* X X) (SIN (IFLTE (%
 Y Y) (% (SIN (SIN (% Y 0.30400002))) X) (% Y 0.30400002) (IFLTE (IFLTE (%
(SIN (% (% Y (+ X Y)) 0.30400002)) (+ X Y)) (% X -0.10399997) (- X Y) (* (+
-0.12499994 -0.15999997) (- X Y))) 0.30400002 (SIN (SIN (IFLTE (% (SIN (%
(% Y 0.30400002) 0.30400002)) (+ X Y)) (% (SIN Y) Y) (SIN (SIN (SIN (% (SIN
X) (+ -0.12499994 -0.15999997))))) (% (+ (+ X Y) (+ Y Y)) 0.30400002)))) (+
(+ X Y) (+ Y Y))))) (SIN (IFLTE (IFLTE Y (+ X Y) (- X Y) (+ Y Y)) (* X X)
(SIN (IFLTE (% Y Y) (% (SIN (SIN (% Y 0.30400002))) X) (% Y 0.30400002)
(SIN (SIN (IFLTE (IFLTE (SIN (% (SIN X) (+ -0.12499994 -0.15999997))) (% X
-0.10399997) (- X Y) (+ X Y)) (SIN (% (SIN X) (+ -0.12499994 -0.15999997)))
(SIN (SIN (% (SIN X) (+ -0.12499994 -0.15999997)))) (+ (+ X Y) (+ Y
Y))))))) (% Y 0.30400002)))))
```

Figure 6 shows the fitness histograms for five different generations of the run. Each bar in the histogram represents a range of ten levels of fitness between 0 and 194. Note the undulating left-to-right movement of the fitness of the population over the generations.







Figures 6 Fitness histograms for generations 0, 12, and 36.

If we retest the best-of-run individual from generation 36 on the two intertwined spirals with sample points chosen twice as dense, we find that 372 of the 388 points (i.e., 96%) are correctly classified. And, if we retest with sample points that are ten times more dense, we find that 1,818 of the 1,940 points (i.e., 94%) are still correctly classified.

Note that we did not pre-specify the size and shape of the solution to the problem. As we proceeded from generation to generation, the size and shape of the best-of-generation individuals changed. The structure of the S-expression emerged as a result of the selective pressure exerted by the fitness measure (i.e. number of fitness cases correctly classified).

6 ACKNOWLEDGMENTS

James P. Rice of the Knowledge Systems Laboratory at Stanford University made numerous contributions in connection with the computer programming of the above.

7 REFERENCES

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

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

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

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.

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

Koza, John R. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Stanford University Computer Science Department Technical Report STAN-CS-90-1314. June 1990.

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: MIT Press 1991.Pages 366-375. 1991a.

Koza, John R. Concept formation and decision tree induction using the genetic programming paradigm. In Schwefel, Hans-Paul and Maenner, Reinhard (editors). *Parallel Problem Solving from Nature.* Berlin: Springer-Verlag. 1991. Pages 124-128. 1991b.

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. 1991c.

Koza, John R. A hierarchical approach to learning the Boolean multiplexer function. In Rawlins 1991. Pages 171-192. 1991d.

Koza, John R. Evolving a computer program to generate random numbers using the genetic programming paradigm. In Belew and Booker 1991. Pages 37-44. 1991e.

Koza, John R. *Genetic Programming*. MIT Press, Cambridge, MA, 1992a.

Koza, John R. Evolution of subsumption using genetic programming. In Bourgine, Paul and Varela, Francisco (editors). *Proceedings of European Conference on Artificial Life, Paris, December 1991.* Cambridge, MA: MIT Press 1992b.

Koza, John R. and Keane, Martin A. Cart centering and broom balancing by genetically breeding populations of control strategy programs. In *Proceedings of International Joint Conference on Neural Networks*, *Washington, January 15-19, 1990*. Volume I, Pages 198-201. Hillsdale, NJ: Lawrence Erlbaum 1990. 1990a.

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. 1990b.

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.* IEEE Press. 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*. Redwood City, CA: Addison-Wesley 1991b.

Lang, Kevin J. and Witbrock, Michael J. Learning to tell two spirals apart. *Proceedings of the 1988 Connectionist Models Summer School*. San Mateo, CA: Morgan Kaufmann 1989. Pages 52-59.

Nguyen, Derrick and Widrow, Bernard. The truck backer-upper: An example of self-learning in neural networks. In Miller, W. Thomas III, Sutton, Richard S., and Werbos, Paul J. (editors). *Neural Networks for Control*. Cambridge, MA: MIT Press 1990.

Rawlins, Gregory (editor). *Proceedings of Workshop on the Foundations of Genetic Algorithms and Classifier Systems*. *Bloomington, Indiana. July 15-18, 1990.* San Mateo, CA: Morgan Kaufmann 1991.