Presented July 15, 1992 to the International Conference on Game Theory and Its Applications held at Stony Brook, New York.

GENETIC EVOLUTION AND CO-EVOLUTION OF GAME STRATEGIES

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

ABSTRACT

The problem of discovering a strategy for playing a game can be viewed as requiring discovery of a computer program. This paper describes the recently developed "genetic programming" paradigm which genetically breeds populations of hierarchical computer programs to solve problems. The computer programs are evolved using the Darwinian principle of survival of the fittest and the genetic operation of sexual recombination.

1 INTRODUCTION AND OVERVIEW

The problem of discovering a strategy for playing a game is an important problem in game theory. This problem can be viewed as requiring discovery of a computer program. The desired computer program takes either the entire history of past moves in the game or the current state of the game as its input and produces the next move as its output.

This paper describes the recently developed "genetic programming" paradigm which genetically breeds populations of computer programs to solve problems. In genetic programming, the individuals in the population are independently acting hierarchical compositions of functions and arguments of various sizes and shapes. Each of these individual computer programs is evaluated for its fitness in handling the problem environment. A simulated evolutionary process driven by this measure of fitness then uses the Darwinian principle of reproduction and survival of the fittest and the genetic operation of crossover (sexual recombination) to solve the problem.

The genetic programming paradigm can also operate simultaneously on two (or more) populations of programs. In such "co-evolution," each population acts as the environment for the other population. In particular, each individual of the first population is evaluated for "relative fitness" by testing it against each individual in the second population, and, simultaneously, each individual in the second population is evaluated for "relative fitness" by testing it against each individual in the first population. Over a period of many generations, individuals with high "absolute fitness" may evolve as the two populations mutually bootstrap each other to increasingly high levels of fitness.

In this paper, the genetic programming paradigm is illustrated with three different problems from game theory.

- The first problem involves genetically breeding a population of computer programs to find an optimal strategy for a player of a discrete two-person 32-outcome game represented by a game tree in extensive form. In this problem, the entire history of past moves of both players is used as input to the computer program.
- The second problem involves genetically breeding a minimax control strategy in a differential game with an independently-acting pursuer and evader. In this problem, the state of the game is used as input to the computer program.
- The third problem illustrates the "co-evolution" and involves genetically breeding an optimal strategy for a player of a discrete two-person 32-outcome game represented by a game tree in extensive form.

2 BACKGROUND ON GENETIC ALGORITHMS

Genetic algorithms are highly parallel

mathematical algorithms that transform populations of individual mathematical objects (typically fixed-length binary character strings) into new populations using operations patterned after natural genetic operations such as sexual recombination (crossover) and fitness proportionate reproduction (Darwinian survival of the fittest).

Genetic algorithms begin with an initial population of individuals (typically randomly generated) and then iteratively (1) evaluate the individuals in the population for fitness with respect to the problem environment and (2) perform genetic operations on various individuals in the population to produce a new population.

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.

Recent work in genetic algorithms and genetic classifier systems can be surveyed in Goldberg (1989), Davis (1987), and Schaffer (1989).

3 BACKGROUND ON GENETIC PROGRAMMING

Representation is a key issue in genetic algorithm work because genetic algorithms directly manipulate the coded representation of the problem and because the representation scheme can severely limit the window by which the system observes its world. Fixed length character strings present difficulties for some problems — particularly problems where the desired solution is hierarchical and where the size and shape of the solution is unknown in advance. The need for more powerful representations has been recognized for some time (De Jong 1985, 1988).

The structure of the individual mathematical objects that are manipulated by the genetic algorithm can be more complex than the fixed length character strings first described by Holland (1975) in 1975. Steven Smith (1980) departed from the early fixed-length character strings by introducing variable length strings, specifically, strings whose elements were if-then rules, rather than single characters. Holland's introduction of the genetic classifier system (1986) continued the trend towards increasing the complexity of the structures undergoing adaptation. The classifier system is a cognitive architecture containing a population of string-based if-then rules (whose condition and action parts are fixed length binary strings) which can be modified by the genetic algorithm.

The recently developed genetic programming paradigm further continues the above trend towards increasing the complexity of the structures undergoing adaptation. In the genetic programming paradigm, the individuals in the population are hierarchical compositions of functions and terminals appropriate to the particular problem domain. The hierarchies are of various sizes and shapes. The set of functions typically includes arithmetic operations, mathematical functions, conditional logical operations, and domain-specific functions. Each function in the function set should be well defined for any element in the range of every other function in the set. The set of terminals used typically includes inputs (sensors) appropriate to the problem domain and various constants. The search space is the hyperspace of all possible compositions of functions and terminals that can be recursively composed of 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 basic genetic operations for the genetic programming paradigm are fitness based reproduction and crossover (recombination).

Fitness proportionate reproduction is the basic engine of Darwinian reproduction and survival of the fittest. It copies individuals with probability proportionate to fitness from one generation of the population into the next generation. In this respect, it operates for the genetic programming paradigm in the same way as it does for conventional genetic algorithms. The crossover operation for the genetic programming paradigm is a sexual operation that operates on two parental LISP S-expressions and produces two offspring S-expressions using parts of each parent. Typically the two parents are hierarchical compositions of functions of different size and shape. In particular, the crossover operation starts by selecting a random crossover point in each parent and then creates two new offspring S-expressions by exchanging the sub-trees (i.e. sub-lists) between the two parents. Because entire sub-trees are swapped, this genetic crossover (recombination) operation produces syntactically and semantically valid LISP S-expressions as offspring regardless of which point is selected in either parent.

For example, consider the parental LISP S-expression:

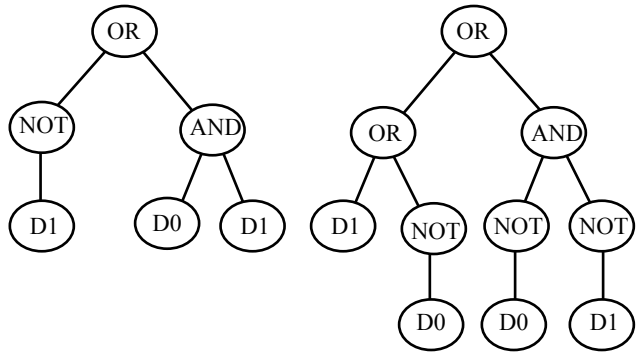
```
(OR (NOT D1) (AND D0 D1))
```

And, consider the second parental S-expression below:

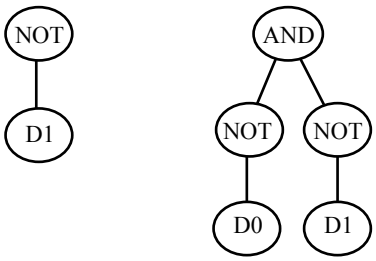
```
(OR (OR D1 (NOT D0))
    (AND (NOT D0) (NOT D1)))
```

These two LISP S-expressions can be depicted graphically as rooted, point-labeled trees with ordered branches. Assume that the points of both trees are numbered in a depth-first way starting at the left. Suppose that the second point (out of 6 points of the first parent) is randomly selected as the crossover point for the first parent and that the sixth point (out of 10 points of the second parent) is randomly selected as the crossover point of the second parent. The crossover points are therefore the NOT in the first parent and the AND in the second parent.

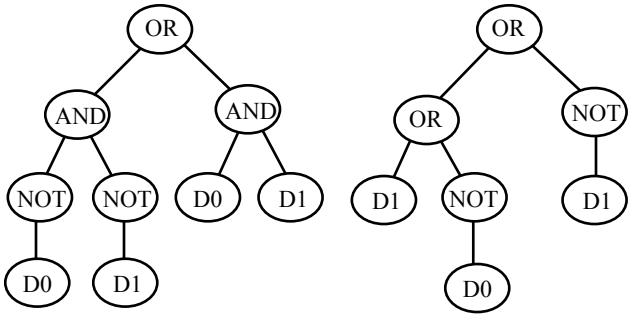
The two parental LISP S-expressions are shown below:



The two crossover fragments are two sub-trees shown below:



These two crossover fragments correspond to the bold, underlined sub-expressions (sub-lists) in the two parental LISP S-expressions shown above. The two offspring resulting from crossover are shown below.



Note that the first offspring above is a perfect solution for the exclusive-or function, namely `(OR (AND (NOT D0) (NOT D1)) (AND D0 D1))`.

Details can be found in Koza (1990a, 1992a, 1992b).

We have shown that entire computer programs can be genetically bred to solve problems in a variety of different areas of artificial intelligence, machine learning, and symbolic processing (1989, 1990a, 1992a, 1992b). In particular, this new paradigm has been successfully applied to example problems in several different areas, including

- automatic programming (e.g. discovering a

computational procedure for solving pairs of linear equations, solving quadratic equations for complex roots, and discovering trigonometric identities),

- empirical discovery (e.g. rediscovering Kepler's Third Law, rediscovering the well-known 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), (Koza 1990b),
- planning (e.g. navigating an artificial ant along an irregular trail, developing a robotic action sequence that can stack an arbitrary initial configuration of blocks into a specified order),
- machine learning of functions (e.g. learning the Boolean 11-multiplexer function),
- sequence induction (e.g. inducing a recursive computational procedure for generating sequences such as the Fibonacci and the Hofstadter sequences),
- symbolic "data to function" regression, symbolic "data to function" integration, and symbolic "data to function" differentiation, and
- symbolic solution to functional equations (including differential equations with initial conditions, integral equations, and general functional equations).

The genetic programming paradigm permits the evolution of computer programs which can perform alternative computations conditioned on the outcome of intermediate calculations, which can perform computations on variables of many different types, which can perform iterations and recursions to achieve the desired result, which can define and subsequently use computed values and sub-programs, and whose size, shape, and complexity is not specified in advance.

4 MINIMAX STRATEGY FOR A SIMPLE DISCRETE GAME

As a first illustration of the genetic programming paradigm, consider the discrete game whose game tree is presented in extensive form in Figure 1.

This game is a two-person, competitive, zero-sum game in which the players make alternating moves. On each move, a player can choose to go L (left) or R (right). Each internal point of this tree is labeled with the player who must move. Each line is labeled with the choice (either L or R) made by the moving player. Each endpoint of the tree is labeled with the payoff (to player X). After player X has made three moves and player O has made two moves, player X receives (and player O pays out) the particular payoff shown at the particular endpoint of the game tree.

Since this 32-outcome discrete game is a game of complete information, each player has access to complete information about his

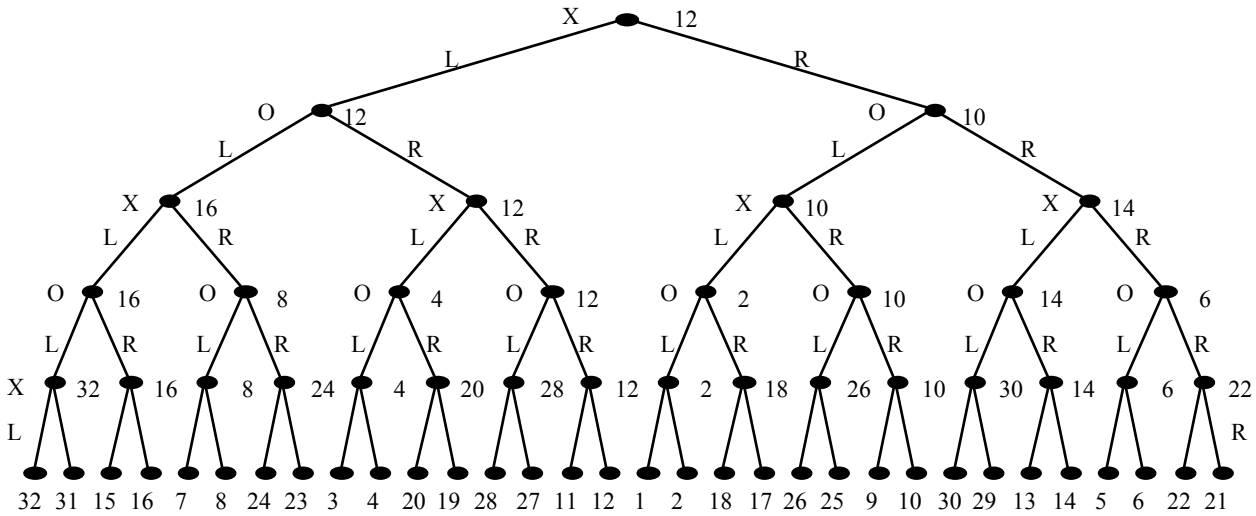


Figure 1 Game Tree with Payoffs

opponent's previous moves and his own previous moves. This information is contained in four variables XM1 (X's move 1), OM1 (O's move 1), XM2 (X's move 2), and OM2 (O's move 2). These variables each assume one of three possible values: L (left), R (right), or U (undefined). A variable is undefined (U) prior to the time when the move to which it refers has been made. Thus, at the beginning of the game, all four variables are undefined. The particular variables that are defined and undefined at a particular time can be used to indicate the point to which play has progressed in the game. For example, if both players have moved once, XM1 and OM1 are both defined (each being either L or R) but the other two variables (XM2, and OM2) are still undefined (i.e., have the value U).

A strategy for a particular player in a game specifies which choice that player is to make for every possible situation that may arise for that player. For this particular game, a strategy for player X must specify his first move if the game is just beginning. Second, a strategy for player X must specify his second move if player O has already made exactly one move. Third, a strategy for player X must specify his third move if player O has already made exactly two moves.

Since player X moves first, his first move is not conditioned on any previous move. But player X's second move will depend on player O's first move (i.e., OM1), and in general it will also depend on his own first move (XM1). Similarly, player X's third move will depend on player O's first two moves and, in general, his own first two moves.

Similarly, a strategy for player O must specify what choice player O is to make for every possible situation that may arise for player O.

A strategy is a computer program whose inputs are the relevant historical variables (XM1, OM1, XM2, and OM2) and whose output is a move (L or R) for the player involved.

The testing functions CXM1, COM1, CXM2, and COM2 provide the ability to test each historical variable (XM1, OM1, XM2, and OM2) that is relevant to deciding a player's move. Each of these functions is a specialized form of the CASE function in LISP. In particular, for example, the function CXM1 has three arguments. It evaluates its first argument if XM1 (X's move 1) is undefined; it evaluates its second argument if XM1 is L (Left); and it evaluates its third argument if XM1 is R (Right). The functions CXM2, COM1, and COM2 are similarly defined.

The terminal set for this problem is

```
T = {L, R}.
```

The function set for this problem is

```
F = {CXM1, COM1, CXM2, COM2},
```

each taking three arguments.

A typical S-expression for this problem therefore consists of a composition of the four testing functions just described and the two terminals L or R. The value returned by such an S-expression at a given time during the play of the game is the terminal (L or R) found at the endpoint of the tree that is reached by virtue of the actual moves that have been made in the game at that time.

The raw fitness of a particular strategy for a particular player is the sum of the payoffs received when that strategy is played against all possible sequences of combinations of moves by the opposing player. Note that the two players of this particular game make different numbers of moves.

Thus, when we compute the fitness of an X strategy, we must test the X strategy against all four possible combinations of O moves, namely O choosing L or R for moves 1 and 2. Similarly, when we compute the fitness of an O strategy, we must test it against all eight possible combinations of X moves, namely X choosing L or R for moves 1, 2, and 3.

When two minimax strategies are played against each other, the payoff is the value of this game (i.e., 12 for this particular game). A minimax strategy takes advantage of non-minimax play by the other player.

A hit for this problem is the number of fitness cases (out of four for player X or eight for player O) where the strategy being tested achieves a payoff at least as good as that achieved by the

minimax strategy.

We now proceed to evolve a game playing strategy for player X for this game.

In one run, the best-of-generation individual game playing strategy for player X in generation 6 had a raw fitness of 88 and scored four hits:

```
(COM2 (COM1 (COM1 L (CXM2 R (COM2 L L L)
                            (CXM1 L R L))
                    (CXM1 L L R)) L R)
      L (COM1 L R R)).
```

This strategy for player X simplifies to

```
(COM2 (COM1 L L R) L R).
```

Note that this strategy for player X is a composition of the four functions (CXM1, COM1, CXM2, COM2) and two terminals (L and R) and that it returns a value of either L or R.

The interpretation of this best-of-run strategy for player X is as follows. If both OM2 (O's move 2) and OM1 (O's move 1) are undefined (U), it must be player X's first move. That is, we are at the beginning of the game (i.e., the root of the game tree). In this situation, the first argument of the COM1 function embedded inside the COM2 function of this strategy specifies that player X is to move L. The left move by player X at the beginning of the game is player X's minimax move because it takes the game to a point with a minimax value of 12 (to player X) rather than to a point with a minimax value of only 10.

If OM2 (O's move 2) is undefined but OM1 is defined, it must be player X's second move. In this situation, this best-of-run strategy specifies that player X moves L if OM1 (O's move 1) was L and player X moves R if OM1 was R. If OM1 (O's move 1) was L, player O has moved to a point with a minimax value of 16. Player X should then move L (rather than R) because that move will take the game to a point with a minimax value of 16 (rather than 8). If OM1 was R, player O has moved to a point with minimax value 12. This move is better for O than moving L. Player X should then move R (rather than L) because that move will take the game to a point with a minimax value of 12 (rather than 4).

If both OM1 and OM2 are defined, it must be player X's third move. If OM2 was L, player X can either choose between a payoff of 32 or 31 or between a payoff of 28 or 27. In either case, player X moves L. If OM2 was R, player X can choose between a payoff of 15 or 16 or between a payoff of 11 or 12. In either case, player X moves R. In this situation, this best-of-run S-expression specifies that player X moves L if OM2 (O's move 2) was L and player X moves R if OM2 was R.

If player O has been playing his minimax strategy, this best-of-run S-expression for player X will cause the game to finish at the endpoint with the payoff of 12 to player X. However, if player O was not playing his minimax strategy, this strategy will cause the game to finish with a payoff of 32, 16, or 28 for player X. The total of 12, 32, 16, and 28 is 88. The attainment of these four values for player X (each at least as good as the minimax value of 12) constitutes four hits for player X.

We used a similar method to evolve a game playing strategy for player O for this game.

In one run, the best-of-generation individual strategy for player O in generation 9 had a raw fitness of 52 and scored eight hits and was, in fact, the minimax strategy for player O:

```
(CXM2 (CXM1 L (COM1 R L L) L) (COM1 R L (CXM2
L L R))
      (COM1 L R (CXM2 R (COM1 L L R) (COM1 R
L R)))).
```

This strategy for player O simplifies to

```
(CXM2 (CXM1 $ R L) L R),
```

where the $ denotes a portion of an S-expression that is inaccessible by virtue of unsatisfiable conditions.

5 DIFFERENTIAL PURSUIT GAME

As a second illustration of genetic programming involving games, consider a differential pursuer-evader game. In particular, consider the "game of simple pursuit" described in Isaacs' *Differential Games* (1965) in which the goal is to find a minimax strategy for one player when playing against a minimax opponent.

This differential pursuer-evader game is a two-person, competitive, zero-sum, simultaneous-move, complete-information game in which a fast pursuing player P is trying to capture a slower evading player E. The choice available to a player at a given moment consists of choosing a direction (angle) in which to travel. In this game, the players may travel anywhere in a plane and

both players may instantaneously change direction without restriction. Each player travels at a constant speed, and the pursuing player's speed w_p (1.0) is greater than the evading player's speed w_e (0.67).

The state variables of the game are x_p, y_p, x_e, and y_e representing the coordinate positions (x_p, y_p) and (x_e, y_e) of the pursuer P and evader E in the plane.

Figure 2 shows the pursuer and the evader. At each time step, both players know the positions (state variables) of both players. The choice for each player is to select a value of his control variable (i.e., the angular direction in which to travel). The pursuer's control variable is the angle ϕ (from 0 to 2π radians), and the evader's control variable is the angle ψ. The players choose their respective control variables simultaneously. In the figure, the evader's angle ψ is shown equal to the pursuer's angle ϕ .

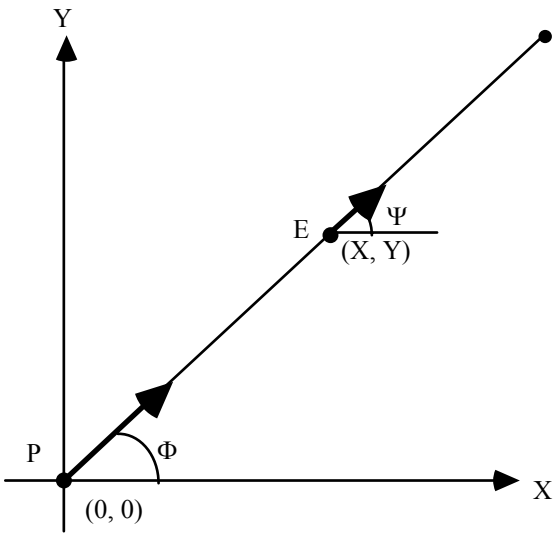


Figure 2 Pursuer P and Evader E.

The analysis of this game can be simplified by reducing the number of state variables from four to two (Isaacs 1965). This state reduction is accomplished by simply viewing the pursuer P as being at the origin point (0,0) of a new coordinate system at all times and then viewing the evader E as being at position (x, y) in this new coordinate system. The two numbers x and y representing the position (x, y) of the evader E thus become the two reduced state variables of the game. Whenever the pursuer P travels in a particular direction, the coordinate system is immediately adjusted so that the pursuer is repositioned to the origin (0, 0). The position (x, y) of the evader is then adjusted to reflect the travel of the pursuer.

The state-transition equations for the evader E are

$$x(t + 1) = x(t) + w_e \text{Cos } \psi - w_p \text{Cos } \phi$$

$$y(t + 1) = y(t) + w_e \text{Sin } \psi - w_p \text{Sin } \phi.$$

We use a set of 20 fitness cases consisting of random initial condition positions (x_i, y_i) for the evader. Each initial condition value of x_i and y_i lies between –5.0 and +5.0. We regard the pursuer as having captured the evader when the pursuer gets to within a small capture radius ε = 0.5 of the evader.

The payoff for a given player is measured by time. The payoff for the pursuer P is the total time it takes to capture the evader E over all the initial condition cases (i.e., fitness cases). The pursuer tries to minimize the time to capture. The payoff for the evader is the total time of survival for E. The evader tries to maximize this time of survival.

A maximum allowed time of 100 time steps is established so that if a particular pursuer strategy has not made the capture within that amount of time, that maximum time becomes the payoff for that particular fitness case and that particular strategy.

The problem is to find the strategy for choosing the control variable of the pursuer so as to minimize the total time to capture for any set of fitness cases when playing against an optimal evader.

For this game, the best strategy for the pursuer P at any given time step is to chase the evader E in the direction of the straight line currently connecting the pursuer to the evader. And, for this game, the best strategy for the evader E is to race away from the pursuer in the direction of the straight line connecting the pursuer to the evader.

In comparison, the worst strategy for the pursuer P is to avoid the evader E by racing away from the evader in the direction precisely opposite to the straight line currently connecting the

pursuer to the evader. The worst strategy for the evader E is to race toward the pursuer P along this same straight line.

If the evader chooses some action other than the strategy of racing away from the pursuer in the direction of the straight line connecting the pursuer to the evader (as shown in figure 3), the evader will survive for less time than if he follows his best strategy. If the evader initially chooses a suboptimal direction and then belatedly chooses the optimal direction, his time of survival is still less than if he had chosen the optimal direction from the beginning.

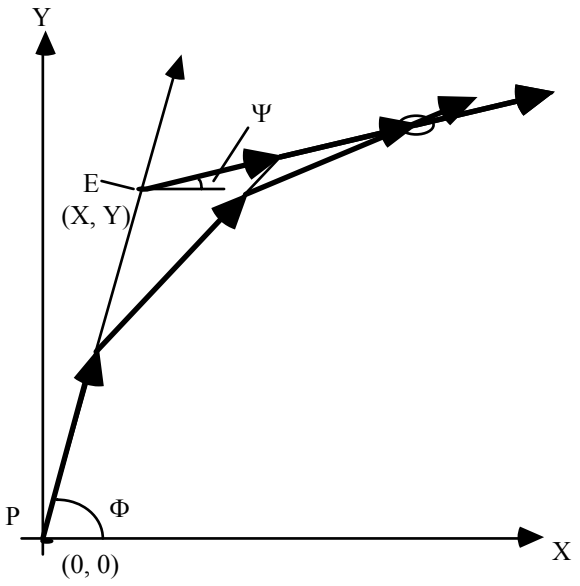


Figure 3 Evader E choosing a suboptimal evasion strategy.

The situation is symmetric in that if the pursuer does not chase after the evader E along the straight line, he fails to minimize the time to capture.

The value of the game is the payoff (time) such that, no matter what the evader does, the evader cannot hold out for longer than this amount of time. If the evader does anything other than direct fleeing, his survival time is a shorter. Conversely, no matter what the pursuer does, the pursuer P cannot capture an optimal evader E in less than that amount of time. And, if the pursuer does anything other than direct pursuit, the evader can remain at large for a longer amount of time.

We start by evolving a minimax pursuer. In doing this, each individual in the population of pursuing individuals is tested against one minimax evader. The optimal evader travels with the established constant evader speed w_e in the angular direction specified by the two argument Arctangent function (which is able to return an angle in the correct quadrant since it can examine the signs of the two arguments).

We later, separately, evolve a minimax evader. Each individual in the population of evading individuals is tested against the minimax pursuer.

The terminal set for this problem consists of the two state variables X and Y representing the position of the evader E in the plane in a reduced coordinate system where the pursuer is always positioned (or repositioned) at the origin and the ephemeral random floating-point constant ← ranging between –1.000 and +1.000 as shown below:

```
T = {X, Y, ←}.
```

The function set for this problem is

```
F = {+,  -, *, %, EXP, IFLTZ}.
```

Note that we did not include any trigonometric or inverse trigonometric function in this function set. Instead, we included the four arithmetic functions, the exponential function, and the three argument conditional operation `IFLTZ` (If Less Than Zero) for making decisions.

For any given S-expression composed of functions from this function set and terminals from this terminal set and any given current position (*x*,*y*) of the pursuer, the S-expression will evaluate to a number that provides the new direction of motion, in radians, for the pursuer.

The fitness cases for this problem consist of 20 initial condition points (*x*,*y*) in a square whose opposite corners are (–5.0, –5.0) and (+5.0, +5.0). The raw fitness for this problem is the average time to capture for each of the fitness cases. The shorter the time, the better.

As one progresses from generation to generation, the population of pursuing individuals typically improves. In early generations, the best pursuing individual in the population can capture

the evader in only a fraction of the 20 fitness cases within the allotted time. These individuals typically do not move in the 100%-efficient straight line called for by the Arctangent strategy, but instead follow a leisurely curved nonoptimal trajectory. Then, after additional generations, the best pursuing individuals in the population can capture the evader in a larger fraction of the fitness cases and within a shorter amount of time. Typically, these partially effective pursuers are effective in some identifiable fraction of the plane or at some identifiable range of distances, but ineffective in other parts of the plane or at other distances.

In one run, the population improved to the point where the best-of-generation individual from generation 11 was able to capture the evader in 20 of the 20 fitness cases; however, its time was 196% of the optimal time.

Then, after an additional 37 generations, a pursuer strategy emerged in generation 48 that resulted in the capture of the evader for all 20 of the fitness cases in 100.61% of optimal time. This best-of-run S-expression is shown below:

```
(% (+ (IFLTZ (* X 0.6370001) (+ X X) (IFLTZ -
0.674 Y Y)) (IFLTZ X (+ X Y) (* (IFLTZ (* X
0.6370001) (IFLTZ (* X X) (- X (EXP (- (% Y
Y) (IFLTZ (EXP (* Y Y)) (* (- X 0.12900007) -
0.029999971) (+ -0.796 X))))) Y) (IFLTZ (EXP
(- (% (IFLTZ (* X 0.6370001) (+ X X) (- Y
0.12900007)) (- -0.992 Y)) (IFLTZ (IFLTZ Y Y
X) Y X))) (+ (% Y Y) (IFLTZ X (+ X Y) (+
(IFLTZ (* X 0.6370001) (* Y Y) 0.018000007)
(IFLTZ X (+ X Y) (% (IFLTZ Y Y X) (+ -0.617
X)))))) Y)) -0.029999971))) (- X (* (% (*
(IFLTZ (* X 0.6370001) (+ X X) (IFLTZ (* X X)
(- X (EXP (- (% Y Y) (* X X)))) Y)) (- Y (- X
(% Y 0.8460001)))) X) -0.029999971))).
```

This best-of-run S-expression closely matches the desired Arctangent behavior. A near-optimal evader has been similarly evolved using an optimal pursuer (i.e., the Arctangent strategy).

We can measure the performance of a probabilistic algorithm by estimating the expected number of individuals that need to be processed by the algorithm in order to produce a solution to the given problem with a certain probability (say 99%). Suppose, for example, a particular run of a genetic algorithm produces the desired result with only a probability of success p_S after a specified choice (perhaps arbitrary and non-optimal) of number of generations N_{gen} and population of size N. Suppose also that we are seeking to achieve the desired result with a probability of, say, $z = 1 - \varepsilon = 99\%$. Then, the number K of independent runs required is

$$K = \frac{\log (1-z)}{\log (1-p_S)} = \frac{\log \varepsilon}{\log (1-p_S)}, \text{ where } \varepsilon = 1-z.$$

For example, we ran 111 runs of the differential pursuer-evader game problem with a population of 500 pursuers and found that the probability of success p_S, after 51 generations, was 55% (see graph below). With a probability of success p_S of 55%, K = 6 independent runs are required to assure a 99% probability of solving the problem on at least one of the runs. That is, it is sufficient to process 153,000 individuals to achieve the desired 99% probability of solving the problem.

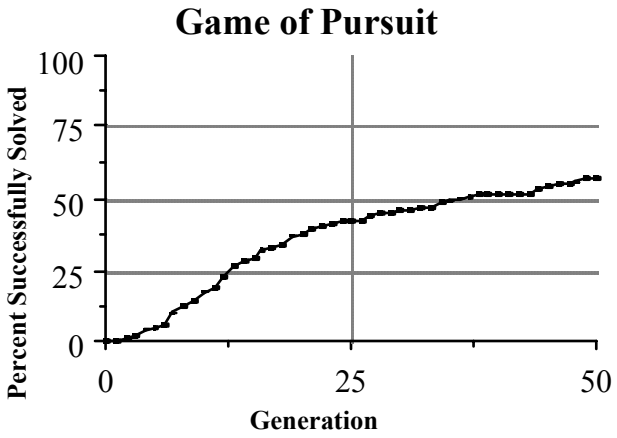


Figure 4 Probability of success p_S by generation.

6 CO-EVOLUTION OF A GAME PLAYING STRATEGY

In the previous section, we genetically bred the strategy for one player in a game by testing each individual in the evolving population of strategies against the minimax strategy for the opposing player or against an exhaustive set of combinations of choices by the opposing player. However, in game theory and in practice, one almost never has *a priori* access to a minimax strategy for the opposing player or the ability to perform an exhaustive test. Since exhaustive testing is practical only for very small games, one faces a situation where genetically breeding a minimax strategy for one player requires already having the minimax strategy for the other player.

Players of checkers or chess know that it is difficult for a new player to learn to play well if he does not have the advantage of playing against a reasonably competent player.

The evolutionary process in nature is often described as if one population of individuals is alone in adapting to a fixed environment; however, this description is only a first order approximation to the actual situation. The environment actually consists of both the physical environment (which may be relatively unchanging) and other independently acting biological populations of individuals which are simultaneously actively adapting to their environment. The actions of each of these other independently acting biological populations (species) usually affect all the other coexisting species. In other words, the environment of a given species includes all the other species that contemporaneously occupy the physical environment and which are simultaneously trying to survive. In biology, the term "co-evolution" is sometimes used to reflect the fact that all species are simultaneously co-evolving in a given physical environment.

A biological example presented by Holland (1990) illustrates the point. A given species of plant may be faced with an environment containing insects that like to eat it. To defend against its predators (and increase its probability of survival in the environment), the plant may, over a period of time, evolve a tough exterior that makes it difficult for the insect to eat it. But, as time passes, the insect may retaliate by evolving a stronger jaw so that the insect population can continue to feed on the plant (and increase its probability of survival in the environment). Then, over an additional period of time, the plant may evolve a poison to help defend itself further against the insects. The insect may subsequently evolve a digestive enzyme that negates the effect of the poison so that the insect population can continue to feed on the plant.

In effect, both the plant and the insects get better and better at their respective defensive and offensive roles in this "biological arms race." Each species changes in response to the actions of the other (Dawkins 1987).

In the basic genetic algorithm described by Holland (1975), a population of individuals attempts to adapt to a fixed environment. The individuals in the population are fixed-length character strings (typically binary strings) that are encoded to represent the problem in some way. In the basic genetic algorithm, the performance of the individuals in the population is measured using a fitness measure which is, in effect, the environment for the population. Over a period of many generations, the genetic algorithm causes the individuals in the population to adapt in a direction that is dictated by the fitness measure (that is, the environment).

In his ECHO system, Holland (1990, 1992) used co-evolution along with a conventional genetic algorithm for exploring the co-evolution of artificial organisms in a "miniature world." Each of the diverse artificial organisms is described by a character string (chromosome). The environment of each organism includes all other organisms.

John Miller (1988, 1989) used co-evolution along with a genetic algorithm to evolve a finite-state automaton as the strategy for playing the repeated prisoner's dilemma game. Miller used a fixed-length character string of 148 binary digits to represent a finite automaton with 16 states. Each automaton, in turn, represented a complete strategy by which to play the game. That is, the automaton specified what move the player was to make for any sequence of previous moves by both players in the game. Miller then used co-evolution to evolve strategies.

Miller's *co-evolutionary* approach to the repeated prisoner's dilemma using the conventional genetic algorithm contrasts with Axelrod's *evolutionary* approach (1987) to the repeated prisoner's dilemma using the conventional genetic algorithm. Axelrod measured the performance of a particular strategy by playing it against a fixed suite of eight superior opposing computer programs which he had selected from those entered into an international programming tournament for the repeated prisoner's dilemma game. In Axelrod's work, fitness was a weighted mix of the results of playing the eight selected opposing computer programs. In other words, the eight selected computer programs served as the environment for

evolving Alexrod's fixed-length character strings.

John Maynard Smith (1989) discussed co-evolution in connection with discovering strategies for games, but without using genetic algorithms. See also Hillis (1990).

In co-evolution, there are two (or more) populations of individuals. The environment for the first population consists of the second population. And, conversely, the environment for the second population consists of the first population.

The co-evolutionary process typically starts with both populations being highly unfit (when measured by an absolute fitness measure). Then, the first population tries to adapt to the environment consisting of the second population. Simultaneously, the second population tries to adapt to the environment consisting of the first population.

This process is carried out by testing the performance of each individual in the first population against each individual (or a sampling of individuals) from the second population. The average performance observed is called the *relative fitness* of that individual, because it represents the performance of that individual relative to the environment consisting of the entire second population. Then, each individual in the second population is tested against each individual (or a sampling of individuals) from the first population. Relative fitness comes from the actual testing of individuals against some or all of the individuals in an opposing population.

Note that this measurement of relative fitness for an individual in co-evolution is not an absolute measure of fitness against an optimal opponent, but merely a relative measure when the individual is tested against the current opposing population. If one population contains boxers who throw only left punches, then an individual whose defensive repertoire contains only of defenses against left punches will have high relative fitness. But this individual would have low absolute fitness when tested against any opponent who knows how to throw both left punches and right punches.

Even when both initial populations are highly unfit (relatively and absolutely), the virtually inevitable variation of the initial random population will mean that some individuals have slightly better relative fitness than others. That means that some individuals in each population have somewhat better performance than others in dealing with the current opposing population.

The operations of crossover and reproduction (based on the Darwinian principle of survival and reproduction of the fittest) can then be separately applied to each population using the relative fitness of each individual in each separate population.

Over a period of time, both populations of individuals will tend to co-evolve and to rise to higher levels of performance as measured in terms of absolute fitness. Both populations do this without the aid of any externally supplied measure of absolute fitness serving as the environment. In the limiting case, both populations of individuals may evolve to a level of performance that equals the absolute optimal fitness. There is, of course, no guarantee that either population will co-evolve to absolute optimal fitness. Co-evolution is a self-organizing, mutually bootstrapping process that is driven only by relative fitness (and not by absolute fitness).

We now illustrate co-evolution by means of genetic programming to simultaneously discover minimax strategies for both players in the same discrete two-person 32-outcome game represented by the game tree in extensive form shown in figure 1.

In co-evolution, we cannot proceed as we did in the previous chapter. We do not have access to the minimax opponent to train the population, as we did with the differential pursuer-evader game nor do we have the ability to exhaustively test each possible combination of choices by the opposing player as we did with the 32-outcome discrete game. Instead, we must breed both populations of players simultaneously. That is, we must simultaneously co-evolve strategies for both players.

Both populations start as random compositions of the same functions and terminals used in the 32-outcome discrete game.

In co-evolution, the relative fitness of a particular strategy in a particular population is the

average of the payoffs that the strategy receives when it is played against fitness cases consisting of each strategy in the opposing population of strategies. Note that the particular strategy is played only once against each strategy in the opposing population. When a particular strategy from the first population is tested against a particular strategy from the opposing population, the outcome is completely determined because, by definition, a strategy specifies a choice for all possible situations. Note that we use the average payoff, rather than the sum of the payoffs, for this particular problem because we envision sampling for larger problems of this type.

In co-evolution, raw fitness is relative fitness. Since raw fitness is defined here in terms of averages, the maximum is 32.

The standardized fitness of an individual strategy is the maximum possible value of raw fitness minus the raw fitness for that strategy.

The absolute fitness of a strategy is used solely for monitoring and descriptive purposes and plays no role in the actual co-evolutionary process. The *absolute fitness* of a particular strategy for a particular player in a game is the payoff received when that strategy is played against the minimax strategy for the opponent. A minimax strategy takes advantage of non-minimax play by the other player. Note that this testing of four or eight combinations does not occur in the computation for relative fitness (i.e., raw fitness).

Hits are the number of fitness cases for which the payoff to an individual strategy equals or exceeds the value of the game (i.e., the result of playing two minimax strategies against each other).

When the two minimax strategies are played against each other, the payoff is the value of this game (i.e., 12 for this game).

In one run involving co-evolution, the individual strategy for player X in generation 0 with the best relative fitness was

```
(COM1 L (COM2 (CXM1 (CXM2 R (CXM2 R R R)
(CXM2 R L R)) L (CXM2 L R (COM2 R R R)))
(COM1 R (COM2 (CXM2 L R L) (COM2 R L L) R)
(COM2 (COM1 R R L) (CXM1 R L R) (CXM1 R L
L))) (CXM1 (COM2 (CXM1 R L L) (CXM2 R R L) R)
R (COM2 L R (CXM1 L L L)))) R).
```

This simplifies to

```
(COM1 L (COM2 L L R) R).
```

This individual has relative fitness of 10.08.

The individual in the initial random population for player O with the best relative fitness was an equally complex expression. It simplifies to

```
(CXM2 R (CXM1 $ L R) (CXM1 $ R L)),
```

where $ denotes a portion of an S-expression which is inaccessible by virtue of unsatisfiable conditions. This individual has relative fitness of 7.57.

Neither the best X individual nor the best O individual from generation 0 reached maximal absolute fitness.

Note that the values of relative fitness for the relative best X individual and the relative best O individual from generation 0 (i.e., 10.08 and the 7.57) are each computed by averaging the payoff from the interaction of the individual involved with all 300 individual strategies in the current opposing population.

In generation 1, the individual strategy for player X with the best relative fitness had relative fitness of 11.28. This individual X strategy is still not a minimax strategy. It does not have the maximal absolute fitness.

In generation 1, the best individual O strategy,

```
(CXM2 (CXM1 R R L) (CXM2 L L (CXM2 R L R))
R),
```

attained relative fitness of 7.18. This O strategy simplifies to

```
(CXM2 (CXM1 $ R L) L R).
```

This best-of-generation individual O strategy from generation 1 is, in fact, a minimax strategy for player O. If it were played against the minimax X strategy, it would score 12 (i.e., the value of the game). This one O individual was the first such O individual to attain this level of performance during this run. In co-evolution, the algorithm does not know that this individual is a minimax strategy for player O. The run merely continues.

Figure 5 graphically depicts the best-of-generation S-expression for player O from generation 1.

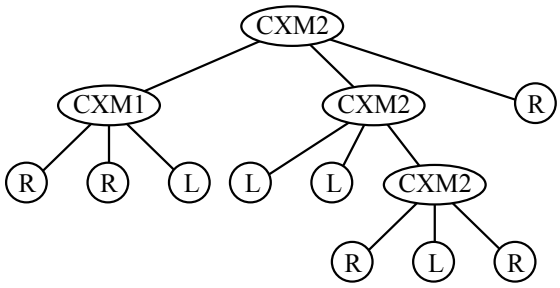


Figure 5 Minimax O strategy from generation 1.

Between generations 2 and 14, the number of individuals in the O population equivalent to the minimax O strategy was 2, 7, 17, 28, 35, 40, 50, 64, 73, 83, 93, 98, and 107, respectively. That is, programs equivalent to the minimax O strategy began to dominate the O population.

In generation 14, the individual strategy for player X with the best relative fitness had a relative fitness of 18.11. This individual X strategy was

```
(COM2 (COM1 L L (CXM1 R R R))
      L
      (CXM1 (COM1 L L (CXM1 R R R))
            (CXM2 L R R) R)).
```

This X strategy simplifies to

```
(COM2 (COM1 L L R) L R).
```

Although the algorithm does not know it, this best-of-generation individual X strategy from generation 14 is, in fact, a minimax strategy for player X. If it were played against the minimax O strategy, it would score 12 (i.e., the value of the game).

Between generations 15 and 29, the number of individuals in the X population equivalent to the minimax X strategy was 3, 4, 8, 11, 10, 9, 13, 21, 24, 29, 43, 32, 52, 48, and 50, respectively. That is, programs equivalent to the minimax X strategy began to dominate the X population. Meanwhile, the O population became even more dominated by programs equivalent to the O minimax strategy.

By generation 38, the number of O individuals in the population reaching maximal absolute fitness reached 188 (almost two thirds of the population) and the number of X individuals reaching maximal absolute fitness reached 74 (about a quarter). That is, by generation 38, the minimax strategies for both players were becoming dominant.

Interestingly, these 74 individual X strategies had relative fitness of 19.11 and the 188 individual O strategies had relative fitness of 10.47. Neither of these values equals 12, because the other population is not fully converged to its minimax strategy.

In summary, we have seen the discovery, via co-evolution, of the minimax strategies for both players in the 32-outcome discrete game. This mutually bootstrapping process found the minimax strategies for both players without using knowledge of the minimax strategy (i.e., any *a priori* knowledge of the game) for either player.

7 CONCLUSIONS

We used the genetic programming paradigm to breed a minimax strategy minimax strategy for a discrete game in extensive form and for a differential game of simple pursuit. We then simultaneously bred an optimal game-playing strategy for both players of a discrete game in extensive form using co-evolution. In co-evolution, two populations are simultaneously co-evolved wherein each population serves as the environment to guide the evolution of the other population.

8 ACKNOWLEDGMENTS

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

9 REFERENCES

Axelrod, R. "The evolution of strategies in the iterated prisoner's dilemma." In *Genetic Algorithms and Simulated Annealing* , edited by L. Davis. London: Pittman l987.

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

Dawkins, Richard. *The Blind Watchmaker*. New York: W. W. Norton 1987.

De Jong, Kenneth A. Genetic algorithms: A l0 year perspective. *Proceedings of an International Conference on Genetic Algorithms and Their Applications.* Hillsdale, NJ: Lawrence Erlbaum Associates l985.

De Jong, Kenneth A. Learning with genetic algorithms: an overview. *Machine Learning,* 3(2), 121-138, 1988.

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

Hillis, W. Daniel. "Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure." In *Emergent Computation: Self-organizing, Collective, and Cooperative Computing Networks*. edited by S. Forrest. Cambridge, MA: MIT Press 1990.

Holland, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press 1975.

Holland, John H. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, Ryszard S., Carbonell, Jaime G. and Mitchell, Tom M. *Machine Learning: An Artificial Intelligence Approach, Volume II.* P. 593-623. Los Altos, CA: Morgan Kaufman l986.

Holland, John H. ECHO: Explorations of evolution in a miniature world. Paper presented at Second Workshop on Artificial Life in Santa Fe, New Mexico, February 1990.

Holland, John H. Second edition of *Adaptation in Natural and Artificial Systems.* Cambridge, MA: The MIT Press 1992.

Isaacs, Rufus.. *Differential Games*. New York: John Wiley 1965.

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: Morgan Kaufman 1989.

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. 1990a.

Koza, John R. "A Genetic Approach to Econometric Modeling." Sixth World Congress of the Econometric Society. Barcelona, Spain. August 27, 1990. 1990b.

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

Miller, J. H. "The Evolution of Automata in the Repeated Prisoner's Dilemma." In *Two Essays on the Economics of Imperfect Information*. PhD dissertation, Department of Economics, University of Michigan, 1988.

Miller, J. H. *The Co-evolution of Automata in the Repeated Prisoner's Dilemma*. Santa Fe Institute Report 89-003. 1989.

Schaffer , J. D. (editor) *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, Ca: Morgan Kaufmann Publishers Inc. 1989.

Smith, John Maynard. *Evolutionary Genetics.* Oxford: Oxford University Press. 1989.

Smith, Steven F. *A Learning System Based on Genetic Adaptive Algorithms*. PhD dissertation. Pittsburgh: University of Pittsburgh 1980.