Submitted September 10, 1991 to 1992 American Control Conference (ACC) to be held in Chicago on June 24-26, 1992

# A Genetic Approach to Finding a Controller to Back Up a Tractor-Trailer Truck

**John R. Koza (Member IEEE, AIAA)**

**Computer Science Department**
**Stanford University**
**Stanford, CA 94305 USA**
**PHONE: 415-941-0336**
**FAX: 415-941-9430**
**Koza@Sunburn.Stanford.Edu**

## ABSTRACT

Problems of control can be viewed as requiring the discovery of a computer program (i.e. control strategy) that takes the state variables of a problem as its inputs and produces the values of the control variables as its output. This paper describes the recently developed genetic programming paradigm which genetically breeds a population of computer programs to solve problems. Genetic programming begins with a population of hundreds or thousands of random computer programs and improves them from generation to generation using the Darwinian operation of fitness proportionate reproduction and the genetic operation of sexual recombination. The sexual recombination operation combines parts of two computer programs, each selected proportional to their fitness, to produce new offspring programs. The paper shows, step by step, how to apply genetic programming to the four dimensional control problem of backing up a tractor-trailer truck to a loading dock. The genetic programming paradigm breeds an approximately correct computer program (i.e. control strategy) that successfully performs the required task.

## 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 four dimensional control problem. In this paper, we use the recently developed genetic programming paradigm to genetically breed a controller for this 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.

## 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.

Genetic algorithms are an efficient way to search a highly non-linear multi-dimensional space. A good overview of the many practical applications of the genetic algorithms operating on fixed length character strings (and other variants of the genetic algorithm) can be found in Goldberg (1989), Davis (1987, 1990), Belew and Booker (1991), and Rawlins (1991)

# 3. BACKGROUND ON GENETIC PROGRAMMING

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

We have shown that computer programs can be genetically bred to solve problems in a surprising variety of different areas. Specifically, the recently developed genetic programming paradigm has been successfully applied to problems in a wide variety of different areas (Koza 1989, 1990, 1991, 1992), including

- discovering inverse kinematic equations (e.g. to move a robot arm to designated target points),
- optimal control (e.g. centering a cart and balancing a broom on a moving cart in minimal time by applying a "bang bang" force to the cart) (Koza and Keane 1990),
- symbolic "data to function" regression, integration, differentiation, and symbolic solution to general functional equations (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),
- planning (e.g. navigating an artificial ant along a trail and developing a robotic plan for stacking blocks in to a desired order),
- 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),
- machine learning of functions (e.g. learning the Boolean 11-multiplexer function),
- automatic programming (e.g. solving pairs of linear equations, solving quadratic equations for complex roots, and discovering trigonometric identities),
- generation of maximal entropy sequences of random numbers,
- pattern recognition (e.g. translation-invariant one-dimensional shape in a linear retina),
- sequence induction (e.g. inducing a recursive procedure for generating sequences such as the Fibonacci and the Hofstadter sequences),
- concept formation and decision tree induction,
- 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.

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 1991).

## 3.1. OBJECTS USED IN GENETIC PROGRAMMING

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 appropriate to the problem domain and various constants. Each function in the function set should be well defined for any combination of elements from the range of every function that it may encounter and every terminal that it may encounter.

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

One can now view the search for a solution to the problem as a search in the hyperspace of all possible compositions of functions that can be recursively composed of the available functions and terminals.

### 3.2. OPERATIONS USED IN GENETIC PROGRAMMING

The basic genetic operations for the genetic programming paradigm are fitness proportionate reproduction and crossover (recombination). The crossover (recombination) operation is a sexual operation that operates on two parental computer programs and produces two offspring programs using parts of each parent. The crossover operation creates new offspring by exchanging sub-trees (i.e. sub-lists) between the two parents. Because entire sub-trees are swapped, this crossover operation always produces syntactically and semantically valid programs as offspring regardless of the crossover points.

For example, consider the two parental computer programs:

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

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

These two programs are depicted as rooted, point-labeled trees with ordered branches in Figure. 1. The numbers appear for reference only.

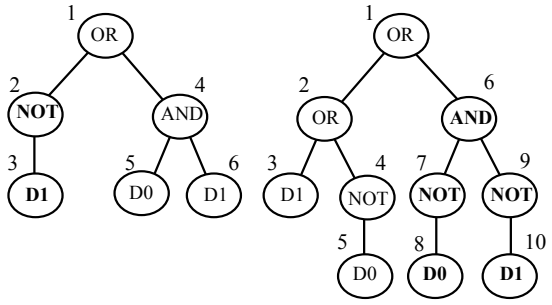


**Figure 1: Two Parental computer programs shown as trees with ordered branches. Internal points of the tree correspond to functions (i.e. operations) and external points correspond to terminals (i.e. input data).**

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

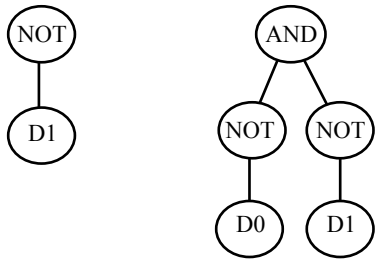


**Figure 2: The Two Crossover Fragments**

These two crossover fragments correspond to the bold, underlined sub-programs (sub-lists) in the two parental computer programs shown above. The two offspring resulting from crossover are shown in Figure 3.

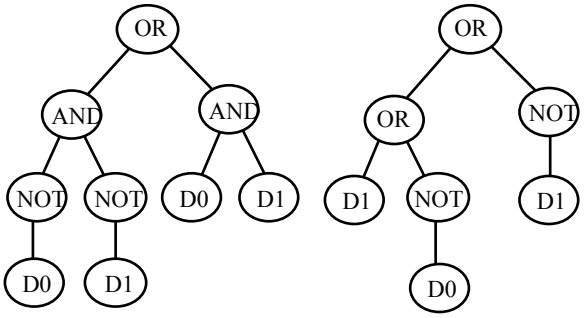


**Figure 3: The Two Offspring Resulting from Crossover**

Note that the first offspring in figure 3 is a computer program for the even-parity (i.e. equal) Boolean function of two arguments (i.e. D0 and D1), namely

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

### 3.3. STEPS REQUIRED TO EXECUTE THE GENETIC PROGRAMMING PARADIGM

The genetic programming paradigm, like the conventional genetic algorithm, is a domain independent method. It proceeds by genetically breeding populations of 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 new computer programs by genetically recombining randomly chosen parts of two existing programs.

(3) The single best computer program in the population at the time of termination is designated as the result of the genetic programming paradigm. This result may be a solution (or approximate solution) to the problem.

## 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.

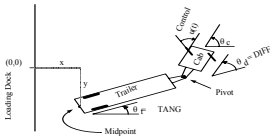


**Figure 4: In the truck backer-upper problem, the goal is to bring the midpoint of the rear of the trailer to the target point (0,0) on the loading dock. The control variable is the steering angle u(t) for the tires of the tractor (cab). The cab is connected to the trailer via the pivot.**

The 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 $\theta t$ (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 $\theta_d$ (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).

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 $\pi$. The length of the tractor (i.e. cab) $d_{\mathrm{C}}$ is 6 meters and the length of the trailer $d_{\mathrm{S}}$ 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 $\theta t$ is TANG. The angle of the tractor relative to the X axis is $\theta_{\mathrm{C}}$.

## 5. PREPARATORY STEPS FOR USING GENETIC PROGRAMMING

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.

We will illustrate these five major steps using the truck backer-upper problem.

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

The first major step in preparing to use 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

$$T = \{X, Y, TANG, DIFF, \leftarrow\},$$

The ephemeral random floating point constant $\leftarrow$ takes on different random floating point values between -1.0 and +1.0 whenever it appears in a computer program in the initial random population (i.e. generation 0).

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 decision function 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 decision function 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.

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 (i.e. X, Y, and TANG attain values which we define as a hit). 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 1000 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 values of the other parameters are the same as we used on all the other problems cited in section 3 to which we have applied the genetic programming paradigm.

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.

## 6. RESULTS

In one run, the best single individual computer program in the initial population of 1000 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 26956. 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)))
```

In just 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 are still ineffective in solving the problem (although their 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 indidivudal 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 an 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.

## 7. CONCLUSIONS

We demonstrated use of the recently developed genetic programming paradigm to genetically breed a control strategy (i.e. computer program) to solve the four dimensional control problem of backing up a tractor-trailer truck to a loading dock.

## 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


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, J. 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.

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. Evolving a computer program to generate random numbers using the genetic programming paradigm. In Belew, Rik and Booker, Lashon (editors) *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann Publishers Inc. 1991.

Koza, John R. *Genetic Programming*. MIT Press, Cambridge, MA, 1992 (forthcoming).

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.

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

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.