Automatic Programming of a Time-Optimal Robot Controller and an Analog Electrical Circuit to Implement the Robot Controller by Means of Genetic Programming

John R. Koza
Computer Science Dept.
Stanford University
Stanford, California 94305
koza@cs.stanford.edu
http://www-cs-faculty.stanford.edu/~koza/

Forrest H Bennett III
Visiting Scholar
Computer Science Dept.
Stanford University
Stanford, California 94305
forrest@evolute.com

Martin A. Keane
Martin Keane Inc.
5733 West Grover
Chicago, Illinois 60630
makeane@ix.netcom.com

David Andre
Computer Science Division
University of California
Berkeley, California
dandre@cs.berkeley.edu

ABSTRACT

Genetic programming is an automatic programming technique that evolves computer programs to solve, or approximately solve, problems. This paper presents two examples in which genetic programming creates a computer program for controlling a robot so that the robot moves to a specified destination point in minimal time. In the first approach, genetic programming evolves a computer program composed of ordinary arithmetic operations and conditional operations to implement a time-optimal control strategy. In the second approach, genetic programming evolves the design of an analog electrical circuit consisting of transistors, diodes, resistors, and power supplies to implement a near-optimal control strategy.

1. Introduction

The solution of non-trivial control problems usually requires execution of momentarily disadvantageous actions in order to ultimately achieve the globally optimal result.

This paper considers the problem of controlling the continuous movement of a robot such that the robot moves to an arbitrary destination point in minimal time. If the robot has a nonzero turning radius, this problem cannot be solved merely by executing a hill-climbing action that greedily improves the distance between the robot and the destination at every intermediate point along the robot's trajectory. Instead, momentarily disadvantageous actions must be taken in the short term in order to achieve the long-term objective.

Section 2 presents the mathematical solution for this time-optimal control problem. Section 3 provides background on genetic programming. Section 4 applies genetic programming to the problem of evolving a control strategy composed of ordinary arithmetic and conditional operations. Section 5 applie genetic programming to evolve a design for an analog electrical circuit to implement a time-optimal control strategy.

2. Statement of the Problem

The problem is to create a strategy for controlling the movement of a robot so that the robot moves to an arbitrary destination point in minimal time.

In figure 1, the robot is initially positioned at the origin (0, 0) of the coordinate system and is headed east (i.e., along the positive x axis). The robot moves at a constant speed, A. The robot's maximum turn angle $\Theta_{\max}$ defines a turning radius, R. The two circles centered at (0, $+R$) and (0, $-R$) in the figure each have a radius equal to this turning radius. The robot's movement is controlled by the turn angle Θ, which controls the robot's change in heading. Angles are measured in radians counterclockwise from the positive x axis.

Suppose the target point lies on the positive x axis, such as point (x_1, y_1) in figure 1. In this first case, the robot's time-optimal control strategy is to move straight ahead (east) along the positive x axis (i.e., with $\Theta = 0$ radians). The time-optimal trajectory is the straight line between the origin and the target (x_1, y_1).

Now suppose that the target point, such as (x_3, y_3), lies on the circumference of the upper circle of radius R. In this second case, the time-optimal control strategy is to turn with a heading equal to the maximum turn angle $\Theta = +\Theta_{max}$. The robot's time-optimal trajectory will be the portion of the circumference of the circle between the origin and the target point (x_3, y_3). By symmetry, the strategy for any point on the lower circle, such as (x_6, y_6), is to turn at the maximum turn angle of $\Theta = -\Theta_{max}$.

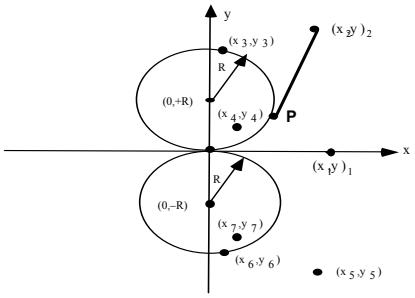

Figure 1 The robot navigation problem.

The remaining points can be classified into two additional cases – outside or inside the circles. The strategies employ portions of the robot trajectories used by one or both of the above strategies.

In the third case, the target point is in the first or second quadrants but *outside* the upper circle. For a point such as (x_2, y_2), there is a unique first point **P** on the circumference of the upper circle such that the tangent line to the upper circle at point **P** passes through (x_2, y_2). For this third case, the minimum-time control strategy for the robot is to turn at the maximum angle $\Theta = +\Theta_{max}$ until the robot reaches point **P** on the circumference of the upper circle and then to move straight ahead (i.e., with an angle of $\Theta = 0$) to the target. The robot's time-optimal trajectory will be the portion of the circumference of the upper circle between the origin and **P** combined with the tangent line between **P** and the target (x_2, y_2). Note that this strategy works regardless of the location of (x_2, y_2) in the first or second quadrants and outside the upper circle (including, specifically, points in the second quadrant behind the robot). By symmetry, a similar strategy (of first turning southeast at the maximum angle of $\Theta = -\Theta_{max}$ and then moving straight) works for target points *outside* the lower circle, such as (x_5, y_5). Points on the negative x axis (i.e., directly behind the robot) can be reached equally well in this manner.

In the fourth case, the target point is in the first or second quadrants but *inside* the upper circle. None of the points inside the upper circle can be reached by pursuing a hill-climbing action that greedily improves the distance between the robot and the target at every intermediate point along the robot's trajectory. Such points can only be reached by incurring a substantial temporary worsening of the distance between the robot and the target. Figure 2 shows, for a point such as (x_4, y_4), that there is a unique point **Q** to the east of the origin on the circumference of the lower circle such that a unique circle is tangent to the lower circle at point **Q** and passes through (x_4, y_4) of the upper circle. For this fourth case, the minimum-time control strategy for the robot is to turn at the maximum angle $\Theta = -\Theta_{max}$ (i.e., initially turning southeast thereby increasing the distance between the robot and the target) until the robot reaches point **Q** and then to reverse directions and turn at the maximum angle $+\Theta_{max}$ until the robot reaches the target. The robot's time-optimal trajectory will be the portion of the circumference of the lower circle between the origin and **Q** and the portion of the circumference of the new tangent circle between **Q** and the target. Note that this strategy works regardless of the location of (x_4, y_4), in the first or second quadrants and inside the upper circle (including, specifically, points behind the robot in the second quadrant). By symmetry, a similar strategy (of first turning northeast at the maximum angle of $\Theta = +\Theta_{max}$ and then turning at the maximum angle of $\Theta = -\Theta_{max}$) works for target points *inside* the lower circle, such as (x_7, y_7).

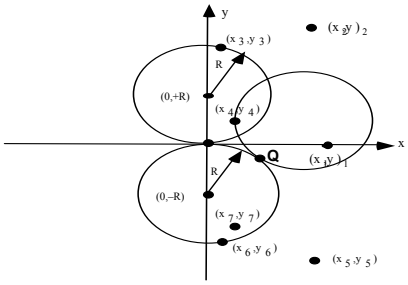

Figure 2 Target point inside upper circle.

The above optimal control strategies and trajectories were described as if the state of the system were the robot's changing position relative to the ground. In practice, the more natural viewpoint for a robot controller is the view from the robot. In this view, whenever the robot travels in a particular direction, the coordinate system is immediately adjusted so that the robot is repositioned to the origin (0, 0) of the coordinate system with the robot heading due east. In this view, the state of the system is the changing location of the target point.

Each of the above four optimal control strategies can be restated in accordance with this new view from the robot. For the first case, the strategy of moving straight ahead has the effect of moving the target point west along the positive x axis until it reaches the robot. For the second case, the strategy of turning with an angle Θ_{max} has the effect of moving the target point around the circumference of the circle until it reaches the robot. For the third case, the strategy has the effect of first rotating

the target to the positive x axis and then moving the target west along the positive x axis until it reaches the robot. For the fourth case, the strategy has the effect of first rotating the target away from the robot to the circumference of the circle and then moving the target point around the circle until it reaches the robot.

The robot has a constant speed, A, of 200 inches per minute and maximum performance rate of turn corresponding to a turn angle of 0.197 radians. The robot is located at the origin of a coordinate system representing a world that extends 4 inches in each of the four directions. A total of 80 time steps of 0.001 minutes each are used to simulate the robot's trajectory. The total simulation time of 0.080 minutes permits a trajectory equivalent to the robot moving twice across its world of 64 square inches.

The fitness of a controller was evaluated using 72 randomly chosen fitness cases each representing a different target point. Fitness was the sum, over the 72 fitness cases, of the travel times. If the robot came within a capture radius of 0.28 inches of its target point before the end of the 80 time steps allowed for a particular fitness case, the contribution to fitness for that fitness case was the actual travel time. However, if the robot failed to come within the capture radius during the 80 time steps, the contribution to fitness for that fitness case was 0.160 minutes (double the worst possible actual travel time).

Calculating fitness requires up to 5,760 executions (72 ∞ 80) of each individual program. For consistency with the later section on designing an electrical circuit where computer time is a major issue, the world of 64 square inches was discretized into a 40 ∞ 40 grid of 1,600 discrete squares (with each square being 0.2 ∞ 0.2 inches). A table was computed for each individual program in the population giving the value of the control variable Θ as if the target point were in each of the 1,600 squares and the robot were at (0,0). When an individual program was executed for a particular one of the 80 time steps of a particular one of the 72 fitness cases, the state of the system was computed from the above state-transition equations using floating-point arithmetic. However, the value of the control variable, Θ, was obtained from the table as if the target were located in the center of its square. In addition, if the robot flew outside of its world of 64 square inches or if the target was moved out of the world relative to the robot's frame of reference, the simulation was terminated and that fitness case was assigned the penalty value of fitness of 0.160 minutes.

3. Genetic Programming

John Holland's pioneering *Adaptation in Natural and Artificial Systems* (1975) described how an analog of the evolutionary process can be applied to solving problems using what is now called the *genetic algorithm*.

The book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Koza 1992) describes an extension of the genetic algorithm in which the genetic population consists of computer programs. See also Koza and Rice 1992. Genetic programming starts with a primordial ooze of randomly generated computer programs composed of the available programmatic ingredients and then applies the Darwinian principle of survival of the fittest and an analog of the naturally-occurring genetic operation of crossover (sexual recombination) to breed a new (and often improved) population of programs.

No approach to automated programming is likely to be successful on non-trivial problems unless it provides some hierarchical mechanism to exploit, *by reuse* and *parameterization*, the regularities, symmetries, homogeneities, similarities, patterns, and modularities inherent in problem environments. Subroutines do this in ordinary computer programs. Accordingly, *Genetic Programming II: Automatic Discovery of Reusable Programs* (Koza 1994a, 1994b) describes how to evolve a multi-part program consisting of a main result-producing branch and one or more reusable, parameterized, hierarchically-called function-defining branches (called *automatically defined functions*).

When automatically defined functions are used, it is necessary to determine the architecture of the yet-to-be-evolved programs. The specification of the architecture consists of (a) the number of function-defining branches in the overall program, (b) the number of arguments (if any) possessed by each function-defining branch, and (c) if there is more than one function-defining branch, the nature of the hierarchical references (if any) allowed between the function-defining branches. This architectural selection can be automated by architecture-altering operations (Koza 1994c) that enable genetic programming to add and delete automatically defined functions and to add arguments and delete arguments of automatically defined functions dynamically during the run.

Recent research on genetic programming is described in Kinnear (1994), Angeline and Kinnear (1996), Koza, Goldberg, Fogel, and Riolo (1996), and Koza et al. (1997).

4. Solution using Arithmetic and Conditional Operations

This section presents the preparatory steps and results for applying genetic programming to the problem of evolving a time-optimal robot control strategy composed of ordinary arithmetic and conditional operations.

Before applying genetic programming to a problem, the user must perform five major preparatory steps,

namely (1) identifying the terminals of the to-be-evolved programs, (2) identifying the primitive functions contained in the to-be-evolved programs, (3) creating the fitness measure, (4) choosing certain control parameters, and (5) determining the termination criterion and method of result designation.

The terminal set for the one result-producing branch, $\mathcal{T}$ of a program in the population for the problem is

$\mathcal{T}$ = {X, Y}.

The function set, $\mathcal{F}$ is

$\mathcal{F}$ = {+, –, *, %, IFLTE, ←}.

The protected division function % takes two arguments and returns one when division by 0 is attempted (including 0 divided by 0), and, otherwise, returns the normal quotient. The four-argument conditional branching operator IFLTE evaluates and returns its third argument if its first argument is less than its second argument, but otherwise evaluates and returns its fourth argument. ← represents floating-point random constants between –1.000 and +1.000.

When a program is executed, it produces a numeric value which the wrapper (output interface) transforms into the turn angle Θ by interpreting the numeric value returned by the program modulo 2π and limiting its absolute value to be less than or equal to Θ max.

Fitness is measured over 72 fitness cases consisting of 72 random destinations (x_i , y_i) for the robot. The x_i and y_i coordinate of each destination lies between –4.0 and +4.0 inches. We regard the robot as having arrived at the designation when it gets to within a capture radius of 0.28 inches of its destination. The fitness is the sum, over the 72 destinations, of the time for the robot to reach its destination. A smaller sum is better.

If the robot has not arrived at its destination within the available 80 time steps (0.080 minutes), the detrimental contribution to fitness for that particular fitness case is double the worst time for a single fitness case (i.e., 0.160 minutes). The worst value of fitness is 11.52 minutes and it occurs when all 72 fitness cases time out in this manner. In comparison, the optimal value of fitness (without the effect of the discretization of the 40 ∞ 40 table) for this problem is known (from Clements 1990) to be 1.518 minutes (an average of about 21 time steps of 0.001 minute each for each of the 72 fitness cases).

The number of hits is defined as the number of fitness cases for which the robot arrives at its destination within the available 80 time steps (i.e., does not time out).

Since there is one output of the to-be-evolved computer program, there is one result-producing branch in each program tree. Accordingly, each program in the initial population of programs (generation 0) has a uniform architecture with no automatically defined functions. After generation 0, the number of automatically defined functions, if any, will emerge as a consequence of the architecture-altering operations (Koza 1994c).

The population size was 40,000. The control parameters were the same as those in Koza 1994a, except (1) The percentage of operations on each generation after generation 5 was 86.5% one-offspring crossovers; 10% reproductions; 1% mutations; 1% branch duplications; 0.5% branch deletions; and 1% branch creations. The percentage of operations before generation 6 was 78% one-offspring crossovers; 10% reproductions; 1% mutations; 5% branch duplications; and 1% branch deletions. (2) The maximum size, H_{rpb}, for the result-producing branch is 400 points. (3) The maximum number of automatically defined functions is 2. (4) The maximum number of arguments for each automatically defined function is two. (5) The maximum size, H_{adf}, for each automatically defined function is 200 points.

This problem was run on a medium-grained parallel Parsytec computer system consisting of four 80 MHz Power PC 601 processors arranged in a loop with a host PC Pentium type computer. The distributed genetic algorithm was used with a population size of Q = 10,000 at each of the D = 4 demes. On each generation, four boatloads of emigrants, each consisting of B = 2% (the migration rate) of the node's subpopulation (selected on the basis of fitness) were dispatched to each of the four toroidally adjacent nodes (Andre and Koza 1996).

4.1. Results

On generation 0 of one run, the best program in the population achieves a fitness of 3.01 and scores 61 hits (out of 72). This program (like all programs of generation 0) has one result-producing branch and no automatically defined functions.

4.1.1. Emergence of Automatically Defined Functions

In subsequent generations, the programs became more complex and their fitness improved. The first pace-setting program containing an automatically defined function appeared in generation 6. This program achieves a fitness of 2.03 and scores 69 hits. The automatically defined function is called only once by the result-producing branch.

4.1.2. Emergence of Reuse

Numerous pace-setting programs with automatically defined functions appear in subsequent generations; however, the first pace-setting program containing a reused automatically defined function appeared in generation 15. This program achieves a fitness of 1.520 and scores 72 hits and calls its ADF0 twice. As it happens, this program is also the first pace-setting program with multiple automatically defined functions; however, its ADF1 is called only once and performs no useful work.

4.1.3. Emergence of Multiple Use of Multiple Automatically Defined Functions

The first pace-setting program containing multiple uses of multiple automatically defined functions appears in generation 25. This program achieves a fitness of 1.454 and scores 72 hits and calls its ADF0 twice and its ADF1 twice. This fitness slightly exceeds the known optimum value because of the granularity of the 40 ∞ 40 grid.

4.1.4. Best-of-Run Individual

The best-of-run program appears in generation 70. This program achieves a fitness of 1.366 and scores 72 hits. It calls its lone one-argument automatically defined function three times. The result-producing branch calls ADF0 four times and ADF1 five times. Figure 3 shows the trajectory for one of the fitness cases in the third quadrant but outside the lower circle, namely the point (–2.58, –2.54).

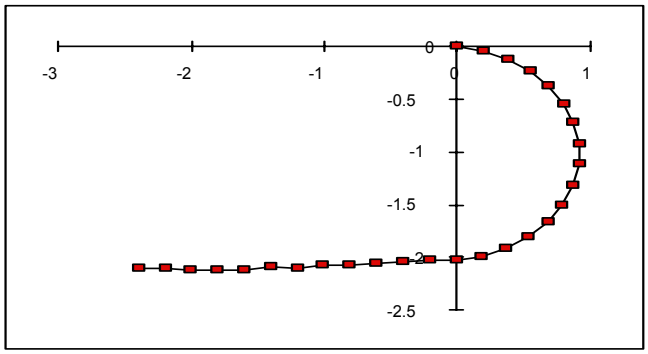

Figure 3 Evolved time-optimal trajectory for fitness case that is outside lower circle.

Figure 4 shows the trajectory for one of the fitness cases in the fourth quadrant but inside the lower circle, namely the point (0.409, –0.892). As can be seen, the trajectory first veers away from the target point, then follows a circular trajectory to the target.

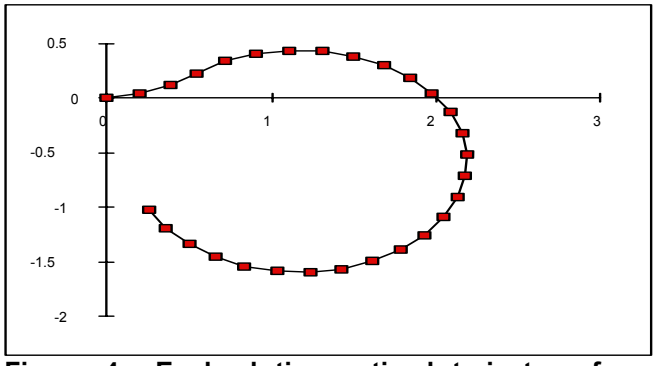

Figure 4 Evolved time-optimal trajectory for fitness case that is outside lower circle.

5. Solution in the Form of an Analog Electrical Circuit

This section presents the preparatory steps and results for the problem of evolving the design of an analog electrical circuit to implement a time-optimal robot controller.

The design of analog circuits is amenable to automation. As Aaserud and Nielsen (1995) observe,

> "Analog designers are few and far between. In contrast to digital design, most of the analog circuits are still handcrafted by the experts or so-called 'zahs' of analog design. The design process is characterized by a combination of experience and intuition and requires a thorough knowledge of the process characteristics and the detailed specifications of the actual product.
>
> "Analog circuit design is known to be a knowledge-intensive, multiphase, iterative task, which usually stretches over a significant period of time and is performed by designers with a large portfolio of skills. It is therefore considered by many to be a form of art rather than a science."

Before applying genetic programming to a problem of circuit synthesis, the user must (in addition to performing the five major preparatory steps described previously) identify an embryonic circuit that is suitable for the problem and specify the method of determining the architecture of the to-be-evolved programs.

The two inputs to the circuit necessary to solve the problem consist of the current location (expressed as two voltages) of the target point in the robot's frame of reference. The output is interpreted by the wrapper (output interface) to be the robot's turn angle Θ. Figure 5 shows a two-input, one-output embryonic circuit.

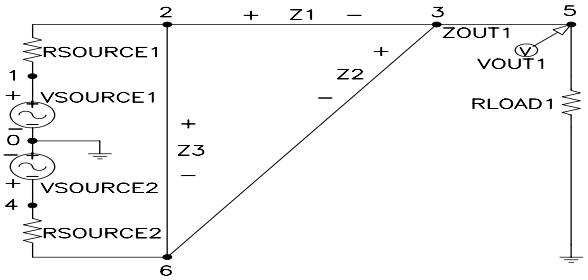

Figure 5 Two-input, one-output embryo.

Since the embryonic circuit has three modifiable wires (Z1, Z2, and Z3), there are three writing heads and three result-producing branches (RPB0, RPB1, RPB2) in each circuit-constructing program tree. The number of automatically defined functions, if any, will emerge as a consequence of the evolutionary process using the architecture-altering operations (Koza 1994c). Each program in the initial population of programs has a uniform architecture with no automatically defined functions (i.e., three result-producing branches).

For this problem, the function set, $\mathcal{F}_{ccs}$, for each construction-continuing subtree is

$\mathcal{F}_{ccs}$ = {R, SERIES, PSS, PSL, FLIP, NOP,
NEW_T_GND_0, NEW_T_GND_1,
NEW_T_POS_0, NEW_T_POS_1,
NEW_T_NEG_0, NEW_T_NEG_1,
PAIR_CONNECT_0, PAIR_CONNECT_1,
Q_D_NPN, Q_D_PNP, Q_3_NPN0, ...,
Q_3_NPN11, Q_3_PNP0, ..., Q_3_PNP11,
Q_POS_COLL_NPN, Q_GND_EMIT_NPN,
Q_NEG_EMIT_NPN, Q_GND_EMIT_PNP,
Q_POS_EMIT_PNP, Q_NEG_COLL_PNP}

Space does not permit a detailed description of each component-creating and connection-modifying function. See Koza, Andre, Bennett, and Keane (1996), and Koza, Bennett, Andre, and Keane (1996a, 1996b, 1997) and Koza, Bennett, Lohn, Dunlap, Andre, and Keane (1997)

The terminal set, $\mathcal{T}$ccs, for the construction-continuing subtree is

$\mathcal{T}$ccs = {END, SAFE_CUT}.

The function set, $\mathcal{F}$aps, for each arithmetic-performing subtree is,

$\mathcal{F}$aps = {+, -}.

The terminal set, $\mathcal{T}$aps, for each arithmetic-performing subtree is

$\mathcal{T}$aps = {←},

where ← represents floating-point random constants from –1.0 to +1.0.

The fitness cases, the fitness measure, the 40 ∞ 40 table, and the wrapper are the same as in the previous section. The fitness calculation starts by executing individual circuit-constructing program tree. The component-creating and connection-modifying functions in the program tree are applied to the embryonic circuit to create a fully developed circuit. For this problem, the voltage VOUT is probed at node 5. The SPICE simulator (Quarles et al. 1994) is requested to perform a nested DC sweep. The nested DC sweep simulates the DC behavior of a circuit with two inputs. A nested DC sweep resembles a nested pair of FOR loops in a computer program in that both of the loops have a starting value for the voltage, an increment, and an ending value for the voltage. For each voltage value in the outer loop, the inner loop simulates the behavior of the circuit by stepping through its range of voltages. Specifically, the starting value for voltage is –4 volts, the step size is 0.2 volts, and the ending value is +4 volts. These values correspond to the dimensions of the robot's world of 64 square inches extending 4 inches in each of the four directions from the origin of a coordinate system (i.e., 1 volt equals 1 inch).

The population size, M, was 640,000. The maximum size, H_{rpb}, for each of the three result-producing branches is 300 points. The maximum number of automatically defined functions is 2. The number of arguments for each automatically defined function is 1. The maximum size, H_{adf}, for each of the automatically defined functions, if any, is 300 points.

5.1. Results

The best circuit from generation 0 (figure 6) scores 61 hits (out of 72) and achieves a fitness of 3.005 and has seven transistors, one diode, and one resistor (not counting the three resistors of the embryo).

The best circuit (figure 7) from generation 17 scores 69 hits and achieves a fitness of 2.06. The best-of-run circuit (figure 8) appeared in generation 31, scores 72 hits, and achieves a near-optimal fitness of 1.541. This circuit has 10 transistors and four resistors.

6. Conclusion

This paper demonstrated the creation, by genetic programming, of a computer program composed of ordinary arithmetic operations and conditional operations for controlling a robot so that the robot moves to a specified destination point in minimal time. In addition, genetic programming evolved the design of an analog electrical circuit consisting of transistors, diodes, resistors, and power supplies that implements a near-optimal strategy.

References

Aaserud, O. and Nielsen, I. Ring. 1995. Trends in current analog design: A panel debate. *Analog Integrated Circuits and Signal Processing*. 7(1) 5-9.

Andre, David and Koza, John R. 1996. Parallel genetic programming: A scalable implementation using the transputer architecture. In Angeline, P. J. and Kinnear, K. E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge: MIT Press.

Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge, MA: The MIT Press.

Clements, John C. 1990. Minimum-time turn trajectories to fly-to points. *Optimal Control Applications and Methods*. 11. Pages 39-50.

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

Kinnear, Kenneth E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: MIT Press.

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

Koza, John R. 1994a. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press.

Koza, John R. 1994c. *Architecture-altering operations for evolving the architecture of a multi-part program in genetic programming*. Stanford University Computer Science Department technical report STAN-CS-TR-94-1528. October 21, 1994.

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin A. 1996. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996a. Four problems for which a computer program evolved by genetic programming is competitive with human performance. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. IEEE Press. Pages 1–10.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996b. Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1997. Evolution using genetic programming of a low-distortion 96 Decibel operational amplifier. *Proceedings of the 1997 ACM Symposium on Applied Computing,* New York: Association for Computing Machinery. 207 - 216.

Koza, John R., Bennett III, Forrest H, Lohn, Jason, Dunlap, Frank, Andre, David, and Keane, Martin A. 1997. Automated synthesis of computational circuits using genetic programming. *Proceedings of the 1997 IEEE Conference on Evolutionary Computation.* Piscataway, NJ: IEEE Press. 447–452.

Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: The MIT Press.

Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). 1997. *Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13–16, 1997, Stanford University*. San Francisco, CA: Morgan Kaufmann.

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

Quarles, Thomas, Newton, A. R., Pederson, D. O., and Sangiovanni-Vincentelli, A. 1994. *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA. March 1994.

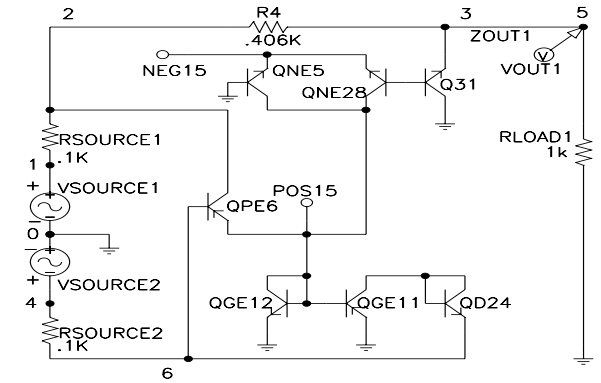

Figure 6 Best circuit of generation 0.

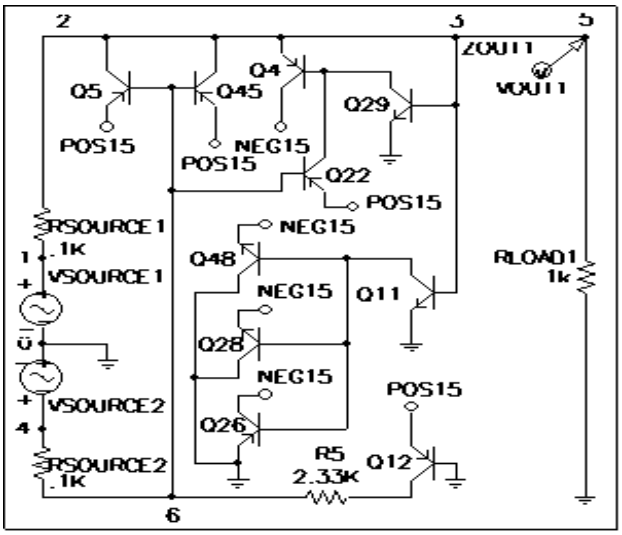

Figure 7 Best circuit of generation 17.

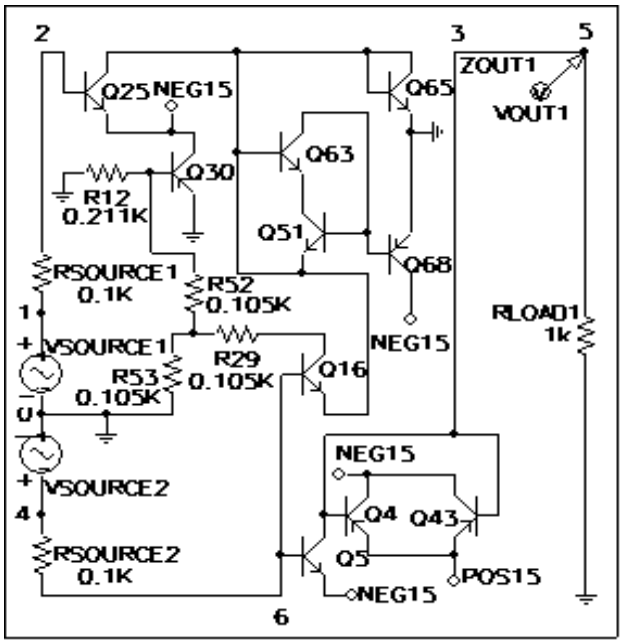

Figure 8 Best-of-run circuit.