Design of a High-Gain Operational Amplifier and Other Circuits by Means of Genetic Programming

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

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

Forrest H Bennett III
Visiting Scholar
Computer Science Dept.
Stanford University
Stanford, California 94305
fhb3@slip.net

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

Abstract: This paper demonstrates that a design for a low-distortion high-gain 96 decibel (64,860 -to-1) operational amplifier (including both circuit topology and component sizing) can be evolved using genetic programming.

1. Introduction

The problem of designing complex structures can be viewed as a search of the space of possible structures for a structure that satisfies the user's requirements and constraints. In nature, complex structures are designed by means of evolution and natural selection.

The possibility of applying the techniques of evolutionary computation to design problems has been recognized by the pioneering work on applying evolutionary programming (EP) to the design of finite automata (Fogel, Owens, and Walsh 1966) and on applying evolution strategies (ES) to airfoil design (Rechenberg 1973). See Fogel, Angeline, and Baeck 1996 for recent work on evolutionary programming and Davidor, Schwefel, Maenner 1994 for recent work on evolutionary strategies.

Holland (1975) described how an analog of the naturally-occurring evolutionary process can be applied to solving scientific and engineering problems using what is now called the *genetic algorithm* (GA). Koza (1992) described an extension of

Holland's genetic algorithm in which the population consists of computer programs. See also Koza and Rice 1992. Koza (1994a, 1994b) describes a way to evolve multi-part programs consisting of a main program and one or more reusable, parameterized, hierarchically-called subprograms (called *automatically defined functions*). Architecture-altering operations provide a way to automatically determine the number of such subprograms, the number of arguments that each possesses, and the nature of the hierarchical references, if any, among such subprograms (Koza 1995). Recent research papers on genetic programming can be found in Kinnear (1994), Angeline and Kinnear (1996), and Koza, Goldberg, Fogel, and Riolo (1996).

The problem of circuit synthesis involves designing an electrical circuit that satisfies user-specified design goals. The design of analog circuits and mixed analog-digital circuits has not proved to be amenable to automation (Aaserud and Nielsen 1995). Thompson (1996) used a genetic algorithm to evolve a frequency discriminator on a Xilinx 6216 reconfigurable gate array in analog mode. CMOS operational amplifier (op amp) circuits have been designed using a version of the genetic algorithm (Kruiskamp and Leenaerts 1995); however, the topology of each op amp was one of 24 topologies based on the conventional human-designed stages of an op amp. Gruau's *cellular encoding* (1996) is an innovative technique in which genetic programming is used to concurrently evolve the architecture, weights, thresholds, and biases of neurons in a neural network.

This paper demonstrates that the design for analog electrical circuits can be evolved using genetic programming. Specifically, designs will be evolved for a low-distortion 96 decibel (dB) op amp (including both the circuit topology and component sizing) as well as several other circuits. The problem-specific information that the user must supply in order to apply genetic programming to a problem of analog circuit synthesis is minimal; it primarily consists of a fitness measure for the operating characteristics of the desired circuit. The user must also specify certain additional basic information such as the number of inputs and outputs of the desired circuit, the set of parts that are to be available to the circuit, and the repertoire of circuit-constructing functions.

2. Development of Circuits from an Embryonic Circuit Using Circuit-Constructing Program Trees

Genetic programming can be applied to circuits if a mapping is established between the kind of rooted, point-labeled trees with ordered branches found in genetic programming and the line-labeled cyclic graphs germane to circuits.

The principles of developmental biology suggest a way to map program trees into circuits. The starting point of the growth process is a very simple embryonic electrical circuit. The embryonic circuit contains certain fixed parts appropriate to the problem at hand and certain wires that are capable of subsequent modification. An electrical circuit is progressively developed by applying the functions in a circuit-constructing program tree to the modifiable wires of the embryonic circuit (and, later, to both the modifiable wires and other components of the successor circuits).

The functions in the circuit-constructing program trees include (1) connection-modifying functions that modify the topology of the circuit, (2) component-creating

functions that insert components into the circuit, (3) arithmetic-performing functions that appear in arithmetic-performing subtrees as argument(s) to the component-creating functions and that specify the numerical value of the component, and possibly (4) calls to automatically defined functions (if used).

Each branch of the program tree is created in accordance with a constrained syntactic structure. Branches are composed from construction-continuing subtrees that continue the developmental process and arithmetic-performing subtrees that determine the numerical value of components. Connection-modifying functions have one or more construction-continuing subtrees, but no arithmetic-performing subtrees. Component-creating functions have one construction-continuing subtree and typically have one arithmetic-performing subtree. This constrained syntactic structure is preserved by using structure-preserving crossover with point typing (Koza 1994a).

The developmental process for converting a program tree into an electrical circuit begins with an embryonic circuit.

Figure 1 shows a one-input, one-output embryonic circuit that serves as a test harness for evolving op amp circuits. VSOURCE is the input signal. VOUT is the output signal. There is a fixed 100 Ohm load resistor RLOAD and a fixed 100 Ohm source resistor RSOURCE. Because we are evolving an amplifier, there is also a fixed 100,000,000 Ohm feedback resistor RFEEDBACK, a fixed 100 Ohm balancing source resistor RBALANCE_SOURCE, and a fixed 100,000,000 Ohm balancing feedback resistor RBALANCE_FEEDBACK. This arrangement limits the possible amplification of the evolving circuit to 1,000,000-to-1 ratio (120 dB).

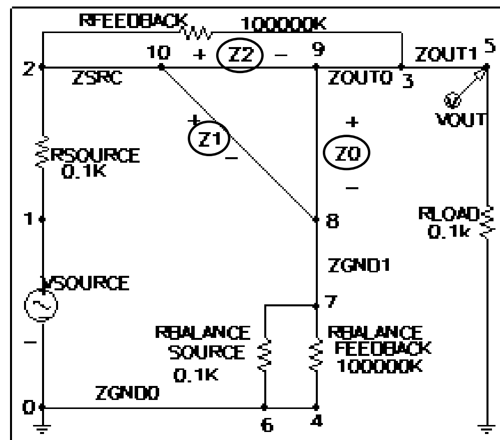

Figure 1 Feedback embryo for an amplifier. All of the above elements (except Z0, Z1, and Z2) are fixed and not modified during the developmental process. At the beginning of the developmental process, there is a writing head pointing to (highlighting) each of the three modifiable wires. All development occurs at wires or components to which a writing head points.

Each circuit-constructing program tree in the population contains component-creating functions and connection-modifying functions. Each connection-modifying function in a program tree points to an associated highlighted component and modifies the topology of the developing circuit. Each branch of the program tree is created in

accordance with a constrained syntactic structure. Branches are composed from construction-continuing subtrees that continue the developmental process and arithmetic-performing subtrees that determine the numerical value of components. Connection-modifying functions have one or more construction-continuing subtrees, but no arithmetic-performing subtrees. Component-creating functions have one construction-continuing subtree and typically have one arithmetic-performing subtree. This constrained syntactic structure is preserved by using structure-preserving crossover with point typing.

Component-creating functions insert a component into the developing circuit and assigns component value(s) to the component. Each component-creating function has a writing head that points to an associated highlighted component in the developing circuit and modifies the highlighted component in a specified way. The construction-continuing subtree of each component-creating function points to a successor function or terminal in the circuit-constructing program tree.

The arithmetic-performing subtree of a component-creating function consists of a composition of arithmetic functions (addition and subtraction) and random constants (in the range –1.000 to +1.000) and specify the numerical value of a component.

Space does not permit a detailed description of each component-creating and connection-modifying function. For details, see Koza, Andre, Bennett, and Keane (1996), and Koza, Bennett, Andre, and Keane (1996a, 1996b, 1996c, 1996d).

3. Preparatory Steps

Our goal is to evolve the design of a high-gain amplifier. Before applying genetic programming to a problem of circuit synthesis, the user must perform seven major preparatory steps, namely (1) identifying the embryonic circuit that is suitable for the problem, (2) determining the architecture of the overall circuit-constructing program trees, (3) identifying the terminals of the to-be-evolved programs, (4) identifying the primitive functions contained in the to-be-evolved programs, (5) creating the fitness measure, (6) choosing certain control parameters (notably population size and the maximum number of generations to be run), and (7) determining the termination criterion and method of result designation.

The feedback embryo for the one-input, one-output amplifier circuit of figure 1 is suitable for this problem.

The embryonic circuit has a writing head associated with each of the three result-producing branches and there are three result-producing branches in each program tree. The number of automatically defined functions, if any, will emerge as a consequence of the evolutionary process using the architecture-altering operations. Each program in the initial population of programs has a uniform architecture with no automatically defined functions (i.e., three result-producing branches).

The terminal sets are identical for all three result-producing branches of the program trees. The function sets are identical for all three result-producing branches.

The initial function set, $\mathcal{F}_{\text{ccs-initial}}$, for each construction-continuing subtree is

$\mathcal{F}_{\text{ccs-initial}}$ = {R, C, 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}

For the npn transistors, the Q2N3904 model was used. For pnp transistors, the Q2N3906 model was used.

The initial terminal set, $\mathcal{T}_{\text{ccs-initial}}$, for each construction-continuing subtree is

$\mathcal{T}_{\text{ccs-initial}}$ = {END, SAFE_CUT}.

The set of potential new functions, $\mathcal{F}_{\text{potential}}$, is

$\mathcal{F}_{\text{potential}}$ = {ADF0, ADF1, ADF2, ADF3}.

The set of potential new terminals, $\mathcal{T}_{\text{potential}}$, is

$\mathcal{T}_{\text{potential}}$ = {ARG0}.

The architecture-altering operations change the function set, $\mathcal{F}_{\text{ccs}}$ for each construction-continuing subtree of all three result-producing branches and the function-defining branches, so

$\mathcal{F}_{\text{ccs}} = \mathcal{F}_{\text{ccs-initial}} \approx \mathcal{F}_{\text{potential}}$.

The terminal set, $\mathcal{T}_{\text{aps}}$, for each arithmetic-performing subtree consists of

$\mathcal{T}_{\text{aps}} = \{\leftarrow\}$,

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

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

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

The evaluation of fitness for each individual circuit-constructing program tree in the population begins with its execution. This execution applies the functions in the program tree to the embryonic circuit, thereby developing the embryonic circuit into a fully developed circuit. A netlist describing the fully developed circuit is then created. The netlist identifies each component of the circuit, the nodes to which that component is connected, and the value of that component. Each circuit is then simulated to determine its behavior. The 217,000-line SPICE (Simulation Program with Integrated Circuit Emphasis) simulation program (Quarles et al. 1994) was modified to run as a submodule within the genetic programming system.

The starting point for evaluating the fitness of a circuit is its response to a DC input. An ideal inverting amplifier circuit would receive a DC input, invert it, and multiply it by the amplification factor. A circuit is flawed to the extent that it does not achieve the desired amplification; to the extent that the output signal is not centered on 0 Volts; and to the extent that the DC response is not linear.

Thus, for this problem, we used a fitness measure based on SPICE's DC sweep. The DC sweep analysis measures the DC response of the circuit at several different DC input voltages. The circuits were analyzed with a 5 point DC sweep ranging from –10 millvolts to +10 mV, with input points at –10 mV, –5 mV, 0 mV, +5 mV, and +10 mV. SPICE then produced the circuit's output for each of these five DC voltages.

Fitness is then calculated from four penalties (i.e., an amplification penalty, a bias penalty, and two non-linearity penalties) derived from these five DC output values.

First, the amplification factor of the circuit is measured by the slope of the straight line between the output for –10 mV and the output for +10 mV (i.e., between the outputs for the endpoints of the DC sweep). If the amplification factor is less than the maximum allowed by the feedback resistor (120 dB for this problem), there is a penalty equal to the shortfall in amplification.

Second, the bias is computed using the DC output associated with a DC input of 0 Volts. The penalty is equal to the bias times a weight. For this problem, a weight of 0.1 is used.

Third, the linearity is measured by the deviation between the slope of each of two line segments and the overall amplification factor of the circuit. The first line segment spans the output values associated with inputs of –10 mv through –5 mv. The second line segment spans the output values associated with inputs of +5 mv and through +10 mv. The penalty for each of these line segments is equal to the absolute value of the difference in slope between the respective line segment and amplification factor of the circuit.

Many of the circuits that are created in the initial random population and many that are created by the crossover and mutation operations cannot be simulated by SPICE. Such circuits are assigned a high penalty value of fitness (10^8).

The population size, M, was 640,000. The architecture-altering operations are used sparingly on each generation. The percentage of operations on each generation after generation 5 was 86.5% one-offspring crossovers; 10% reproductions; 1% mutations; 1% branch duplications; 0% argument duplications; 0.5% branch deletions; 0.0% argument deletions; 1% branch creations; and 0% argument creations. Since we do not want to waste large amounts of computer time in early generations where only a few programs have any automatically functions at all, the percentage of operations on each generation before generation 6 was 78.0% one-offspring crossovers; 10% reproductions; 1% mutations; 5.0% branch duplications; 1% branch deletions; 5.0% branch creations; and 0% argument creations.

The maximum size, H_{rpb}, for each of the three result-producing branches in each overall program is 300 points. The maximum number of automatically defined functions is 4. The number of arguments for each automatically defined function is one. The maximum size, H_{adf}, for each of the automatically defined functions, if any, is 200 points. The other parameters for controlling the runs of genetic programming were the default values specified in Koza 1994a (appendix D).

This problem was run on a medium-grained parallel Parsytec computer system consisting of 64 80 MHz Power PC 601 processors arranged in a toroidal mesh 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 = 64 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 processing nodes. See Andre and Koza 1996 for details.

4. Results for the 96 dB Operational Amplifier

As the run proceeds from generation to generation, the fitness of the best-of-generation individual tends to improve.

The best circuit from generation 50 has 33 transistors, no diodes, eight capacitors, and five resistors (in addition to the five resistors of the feedback embryo). It achieves a fitness of 971,076.4. No automatically defined functions are present in this particular circuit. Figure 2 shows this best circuit from generation 50.

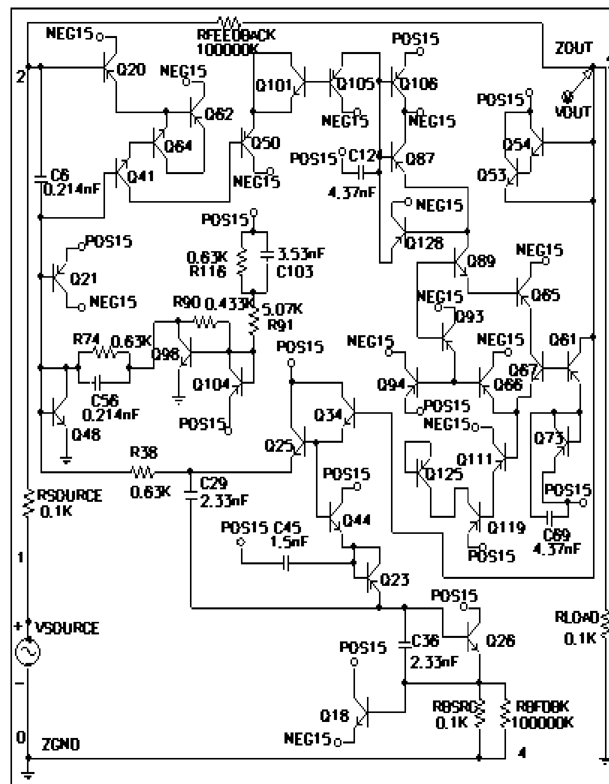

Figure 2 Best circuit from generation 50.

The performance of this best of generation circuit from generation 50 was determined using SPICE. The DC sweep shows that the circuit has an amplification of 89.7 dB (30,545-to-1) and a bias of 9.77 Volts. Based on the time domain behavior for a 20 microvolt sinusoidal 1,000 Hz input signal, the amplification is 89.7 dB (30,500-to-1) for the best circuit from generation 50; the bias is 9.76 Volts; and the distortion is 6.29%. Based on the AC sweep for the best circuit of generation 50, the 3 dB bandwidth is 2,300 Hz. The circuit has a flatband gain is 89.7 dB.

The best-of-run circuit (figure 3) appeared in generation 86 and achieves a fitness of 938,427.3. The program tree has two automatically defined functions. `ADF0` is called once; but `ADF1` is not called. Figure 4 shows automatically defined function `ADF0` of the best circuit from generation 86 (which has 12 transistors, no diodes, one

capacitor, and two resistors). The DC sweep for this best of generation circuit from generation 86 shows that the circuit has an amplification of 96.2 dB (64,860 -to-1) and a bias of 7.44 Volts. Based on the time domain behavior for a 20 microvolt sinusoidal 1,000 Hz signal as input, the amplification is 94.1 for the best circuit from generation 86; the bias is 7.46 Volts; and the distortion is 7.07%. Figure 5 shows the frequency response of this circuit as shown by an AC sweep. The horizontal axis shows frequency on a logarithmic scale from 1 Hz to 1,000,000 Hz. The vertical axis shows gain and ranges from 0 to 100 dB. The 3 dB bandwidth is 1078.4 Hz. The circuit has a flatband gain of 96.3 dB.

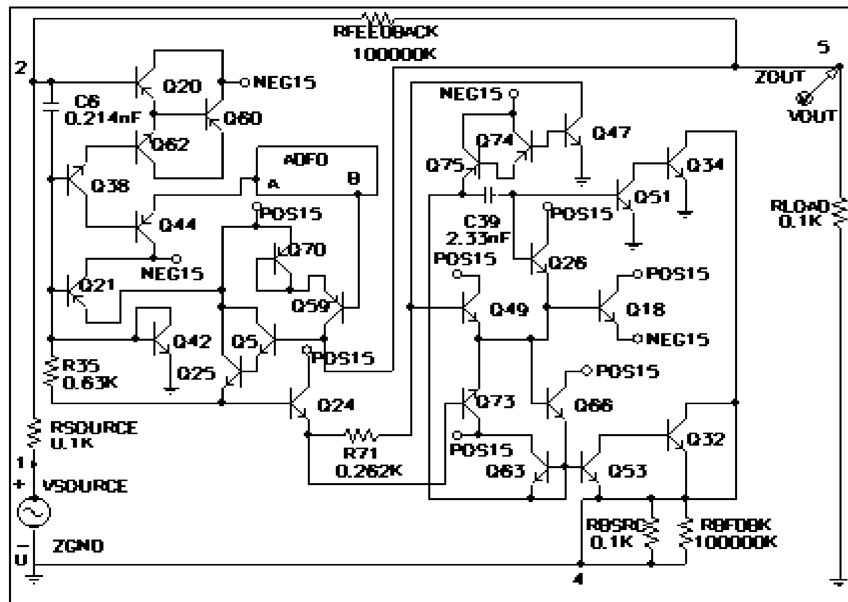

Figure 3 Best circuit from generation 86.

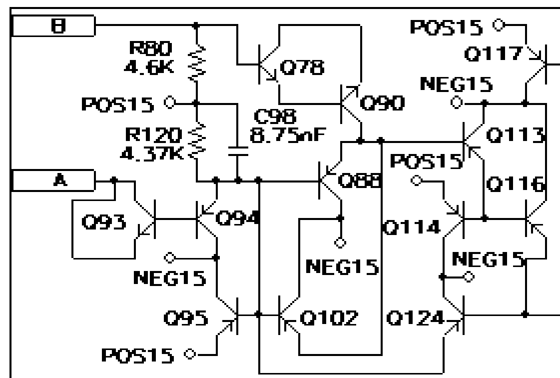

Figure 4 `ADF0` for best circuit from generation 86.

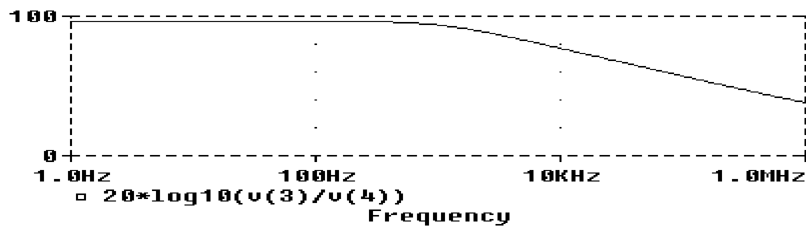

Figure 5 AC sweep for the best circuit from generation 86.

5. Other Circuits Designed with Genetic Programming

Genetic programming has also been successfully applied to a variety of other problems of analog circuit design.

5.1 Lowpass Filter

Genetic programming has successfully evolved a design for a lowpass filter with passband below 1,000 Hz and a stopband above 2,000 Hz with requirements equivalent to that of a fifth order elliptic filter (Koza, Bennett, Andre, and Keane 1996a, 1996c).

Numerous runs produced lowpass filters having a topology that is similar to that employed by human engineers. For example, in one run, a 100% compliant evolved circuit had the recognizable ladder topology of a Butterworth or Chebychev filter (i.e., a composition of series inductors horizontally with capacitors as vertical shunts).

5.2 A Crossover Filter

A design for a crossover (woofer and tweeter) filter was reported in Koza, Bennett, Andre, and Keane 1996b. This problem requires a one-input, two-output embryonic circuit. The lowpass part of the genetically evolved best-of-run circuit has the Butterworth topology. Except for one additional capacitor, the highpass part of this circuit also has the Butterworth topology. This circuit is slightly better than the combination of lowpass and highpass Butterworth filters of order 7.

5.3 Asymmetric Bandpass Filter

A design for an asymmetric bandpass filter with requirements equivalent to a tenth-order elliptic filter was successfully evolved (Koza, Bennett, Andre, and Keane 1996d).

5.4 Cube Root Circuit

Analog electrical circuits that perform mathematical functions (e.g., logarithm, square, cube root) are called *computational circuits*. The design of computational circuits is difficult even for mundane mathematical functions and often relies on the clever exploitation of some aspect of the underlying device physics of the components. Moreover, implementation of each mathematical function typically requires an entirely different clever insight. A design for a computational circuit for the cube root function has been successfully evolved (figure 6).

6. Conclusion

We demonstrated that genetic programming is capable of successfully evolving an op amp circuit that delivers a DC gain of 96 dB (64,860 -to-1).

Acknowledgments

Simon Handley and Jason Lohn made helpful comments on drafts of this paper.

References

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, MA: The MIT Press.

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

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.

Davidor, Yuval, Schwefel, Hans-Paul, and Maenner, Reinhard (editors). 1994. *Parallel Problem Solving from Nature - PPSN III.* Lecture Notes in Computer Science, Volume 866. Berlin: Springer-Verlag.

Fogel, Lawrence J., Owens, Alvin J., and Walsh, Michael. J. 1966. *Artificial Intelligence through Simulated Evolution*. New York: John Wiley.

Fogel, Lawrence J., Angeline, Peter J. and Baeck, T. 1996. *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*. Cambridge, MA: The MIT Press.

Gruau, Frederic. 1996. Artificial cellular development in optimization and compilation. In Sanchez, Eduardo and Tomassini, Marco (editors). 1996. *Towards Evolvable Hardware*. Lecture Notes in Computer Science, Volume 1062. Berlin: Springer-Verlag. Pages 48 – 75.

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: The 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. 1995. Evolving the architecture of a multi-part program in genetic programming using architecture-altering operations. In McDonnell, John R., Reynolds, Robert G., and Fogel, David B. (editors). 1995. *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*. Cambridge, MA: The MIT Press. Pages 695–717.

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: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996a. Toward evolution of electronic animals using genetic programming. *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996b. 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. 1996c. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In Gero, John S. and Sudweeks, Fay (editors). *Artificial Intelligence in Design '96*. Dordrecht: Kluwer. Pages 151-170.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996d. 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: The MIT Press.

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

Kruiskamp M. W. and Leenaerts, D..1995. DARWIN: CMOS opamp synthesis by means of a genetic algorithm. *Proceedings of the 32nd Design Automation Conference*. New York, NY: Association for Computing Machinery. P. 433–438.

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.

Rechenberg, Ingo. 1973. *Evolutionsstrategie*: *Optimierung technischer systeme nach prinzipien der biolgischen evolution*. Stuttgart: Frommann-Holzboog.

Thompson, Adrian. 1996. Silicon evolution. 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.

Figure 6 Genetically evolved cube root circuit.

Version 2 – Camera-Ready – Submitted January 13, 1997 to Sixth Annual Conference on Evolutionary Programming to be held in Indianapolis on April 13 – 16 (Sunday – Wednesday), 1997.

Design of a High-Gain Operational Amplifier and Other Circuits by Means of Genetic Programming

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

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

Forrest H Bennett III
Visiting Scholar
Computer Science Dept.
Stanford University
Stanford, California 94305
fhb3@slip.net

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

NOTE: SLIGHT CHANGE IN TITLE!!!