Version 4 — Submitted ---, 2001 for *Engineering Design Synthesis: Understanding, Approaches and Tools*, edited by: Amaresh Chakrabarti.

Automatic Synthesis of Both the Topology and Numerical Parameters for Complex Structures Using Genetic Programming

John R. Koza
Consulting Professor
Biomedical Informatics
Department of Medicine
Medical School Office Building (MC 5479)
Stanford University
Stanford, California 94305-5479

Consulting Professor
Department of Electrical Engineering
School of Engineering
Stanford University
koza@stanford.edu
http://www.smi.stanford.edu/people/koza

ABSTRACT: This chapter demonstrates that genetic programming can automatically create complex structures from a high-level statement of the structure's purpose. The chapter presents results produced by genetic programming that are from problem areas where there is no known general mathematical technique for automatically creating a satisfactory structure. The results include automatically synthesizing (designing) both the topology (graphical arrangement of components) and sizing (component values) for two illustrative analog electrical circuits and automatically synthesizing both the topology and tuning (component values) for a controller. Genetic programming not only succeeds in producing the required structure, but the structure is competitive with that produced by creative human designers. The claim that genetic programming has produced human-competitive results is supported by the fact that the automatically created results infringe on previously issued patents, improve on previously patented inventions, or duplicate the functionality of previously patented inventions.

1. Introduction

The design of a complex structure typically entails two steps. First, the topology of the structure must be identified. After the topology is identified, optimal (or near-optimal) numerical values can be sought for the elements comprising the structure.

In this chapter, we demonstrate this two-step process with examples involving the design of several types of complex structures.

For example, if one is seeking an analog electrical circuit whose behavior satisfies certain prespecified high-level design goals, one must first ascertain the circuit's topology. Specifically, the *topology* of an electrical circuit comprises

- the total number of electrical components, in the circuit,

- the type of each component (e.g., resistor, capacitor, transistor) at each location in the circuit, and
- a list of all the connections between the leads of the components.

After the topology is established, one must then discover the sizing of each electrical component in the circuit. Specifically, the *sizing* of a circuit consists of the component value(s) for each component of the circuit that requires a component value.

A similar two-step process is required if one is seeking the design of a controller whose behavior satisfies certain prespecified high-level design goals. Controllers are typically composed of signal processing blocks, such as integrators, differentiators, leads, lags, delays, gains, adders, inverters, subtractors, and multipliers (Dorf and Bishop 1998). The purpose of a controller is to force, in a meritorious way, the actual response of a system (conventionally called the *plant*) to match a desired response (called the *reference signal*).

The topology of a controller comprises

- the total number of signal processing blocks in the controller,
- the type of each block (e.g., integrator, differentiator, lead, lag, delay, gain, adder, inverter, subtractor, and multiplier),
- the connections between the inputs and output of each block in the controller and the external input and external output points of the controller.

The *tuning* (sizing) of a controller consists of the parameter values associated with each signal processing block.

A parallel situation arises in connection with networks of chemical reactions (metabolic pathways). The concentrations of substrates, products, and intermediate substances participating in a network of chemical reactions are modeled by non-linear continuous-time differential equations, including various first-order rate laws, second-order rate laws, power laws, and the Michaelis-Menten equations (Voit 2000). The concentrations of catalysts (e.g., enzymes) control the rates of many chemical reactions in living things. The *topology* of a network of chemical reactions comprises

- the total number of reactions,
- the number of substrates consumed by each reaction,
- the number of products produced by each reaction,
- the pathways supplying the substrates (either from external sources or other reactions in the network) to each reaction,
- the pathways dispersing each reaction's products (either to other reactions or external outputs), and
- an identification of whether a particular enzyme acts as a catalyst.

The *sizing* for a network of chemical reactions consists of all the numerical values associated with the network (e.g., the rates of each reaction).

A similarly vexatious situation arises if one is seeking the design of an antenna whose behavior satisfies certain prespecified high-level design goals.

2. Genetic Programming

Genetic programming is an automatic method for solving problems. Specifically, genetic programming progressively breeds a population of computer programs over a series of generations by starting with a primordial ooze of thousands of randomly created computer programs and using the Darwinian principle of natural selection, recombination (crossover), mutation, gene duplication, gene deletion, and certain mechanisms of developmental biology.

Genetic programming breeds computer programs to solve problems by executing the following three steps:

(1) Generate an initial population of compositions (typically random) of the functions and terminals of the problem.

(2) Iteratively perform the following substeps (referred to herein as a generation) on the population of programs until the termination criterion has been satisfied:

(A) Execute each program in the population and assign it a fitness value using the fitness measure.

(B) Create a new population of programs by applying the following operations. The operations are applied to program(s) selected from the population with a probability based on fitness (with reselection allowed).

(i) Reproduction: Copy the selected program to the new population.

(ii) Crossover: Create a new offspring program for the new population by recombining randomly chosen parts of two selected programs.

(iii) Mutation: Create one new offspring program for the new population by randomly mutating a randomly chosen part of the selected program.

(iv) Architecture-altering operations: Select an architecture-altering operation from the available repertoire of such operations and create one new offspring program for the new population by applying the selected architecture-altering operation to the selected program.

(3) Designate the individual program that is identified by result designation (e.g., the best-so-far individual) as the result of the run of genetic programming. This result may be a solution (or an approximate solution) to the problem.

Genetic programming is described in the book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Koza 1992; Koza and Rice 1992), the book *Genetic Programming II: Automatic Discovery of Reusable Programs* (Koza 1994a, 1994b), and the book *Genetic Programming III: Darwinian Invention and Problem Solving* (Koza, Bennett, Andre, and Keane 1999; Koza, Bennett, Andre, Keane, and Brave 1999).

Genetic programming is an extension of the genetic algorithm (Holland 1975) in which the population being bred consists of computer programs.

Genetic programming starts with an initial population of randomly generated computer programs composed of the given primitive functions and terminals. The programs in the population are, in general, of different sizes and shapes. The creation of the initial random population is a blind random search of the space of computer programs composed of the problem's available functions and terminals.

On each generation of a run of genetic programming, each individual in the population of programs is evaluated as to its fitness in solving the problem at hand. The programs in generation 0 of a run almost always have exceedingly poor fitness for non-trivial problems of interest. Nonetheless, some individuals in a population will turn out to be somewhat more fit than others. These differences in performance are then exploited so as to direct the search into promising areas of the search space. The Darwinian principle of reproduction and survival of the fittest is used to probabilistically select, on the basis of fitness, individuals from the population to participate in various operations. A small percentage (e.g., 9%) of the selected individuals are reproduced (copied) from one generation to the next. A very small percentage (e.g. 1%) of the selected individuals are mutated in a random way. Mutation can be viewed as an undirected local search mechanism. The vast majority of the selected individuals (e.g., 90%) participate in the genetic operation of crossover (sexual recombination) in which two offspring programs are created by recombining genetic material from two parents.

The creation of the initial random population and the creation of offspring by the genetic operations are all performed so as to create syntactically valid, executable programs. After the genetic operations are performed on the current generation of the population, the population of offspring (i.e., the new generation) replaces the old generation. The tasks of measuring fitness, Darwinian selection, and genetic operations are then iteratively repeated over many generations. The computer program resulting from this simulated evolutionary process can be the solution to a given problem or a sequence of instructions for constructing the solution.

The dynamic variability of the size and shape of the computer programs that are created during the run is an important feature of genetic programming. It is often difficult and unnatural to try to specify or restrict the size and shape of the eventual solution in advance.

The individual programs that are evolved by genetic programming are typically multi-branch programs consisting of one or more result-producing branches and zero, one, or more automatically defined functions (subroutines).

The *architecture* of such a multi-branch program involves

(1) the total number of automatically defined functions,

(2) the number of arguments (if any) possessed by each automatically defined function, and

(3) if there is more than one automatically defined function in a program, the nature of the hierarchical references (including recursive references), if any, allowed among the automatically defined functions.

Architecture-altering operations enable genetic programming to automatically determine the number of automatically defined functions, the number of arguments that each possesses, and the nature of the hierarchical references, if any, among such automatically defined functions.

Additional information on genetic programming can be found in books such as Banzhaf, Nordin, Keller, and Francone 1998; books in the series on genetic programming from Kluwer Academic Publishers such as Langdon 1998, Ryan 1999, and Wong and Leung 2000; in edited collections of papers such as the *Advances in Genetic Programming* series of books from the MIT Press (Kinnear 1994; Angeline and Kinnear 1996; Spector, Langdon, O'Reilly, and Angeline 1999); in the proceedings of the Genetic Programming Conference held between 1996 and 1998 (Koza, Goldberg, Fogel, and Riolo 1996; Koza, Deb, Dorigo, Fogel, Garzon, Iba, and Riolo 1997; Koza, Banzhaf, Chellapilla, Deb, Dorigo, Fogel, Garzon, Goldberg, Iba, and Riolo 1998); in the proceedings of the annual Genetic and Evolutionary Computation Conference (combining the annual Genetic Programming Conference and the International Conference on Genetic Algorithms) held starting in 1999 (Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith, 1999; Whitley, Goldberg, Cantu-Paz, Spector, Parmee, and Beyer 2000); in the proceedings of the annual Euro-GP conferences held starting in 1998 (Banzhaf, Poli, Schoenauer, and Fogarty 1998; Poli, Nordin, Langdon, and Fogarty 1999; Poli, Banzhaf, Langdon, Miller, Nordin, and Fogarty 2000); at web sites such as `www.genetic-programming.org`; and in the *Genetic Programming and Evolvable Machines* journal (from Kluwer Academic Publishers).

3. Automatic Synthesis of Analog Electrical Circuits

The design process entails creation of a complex structure to satisfy user-defined requirements. The field of design is a good source of problems that can be used for determining whether an automated technique can produce results that are competitive with human-produced results. Design is a major activity of practicing engineers. Since the design process typically entails tradeoffs between competing considerations, the end product of the process is usually a satisfactory and compliant design as opposed to a perfect design. Design is usually viewed as requiring creativity and human intelligence.

The field of design of analog and mixed analog-digital electrical circuits is especially challenging because (prior to genetic programming) there has been no previously known general technique for automatically creating the topology and sizing of an analog circuit from a high-level statement of the circuit's design goals. As Aaserud and Nielsen (1995) noted

> "[M]ost … 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."

Although it might seem difficult or impossible to automatically create both the topology and numerical parameters for a complex structure merely from a high-level statement of the structure's design goals, genetic programming is capable of automatically creating both the topology and sizing (component values) for analog electrical circuits composed of transistors, capacitors, resistors, and other components merely by specifying the circuit's behavior. For purposes of illustrating this point, we discuss

- a lowpass filter circuit using a fitness measure based on the frequency-domain behavior of circuits, and
- a computational circuit employing transistors and using a fitness measure based on the time-domain behavior of circuits.

Genetic programming can be applied to the problem of synthesizing circuits if a mapping is established between the program trees (rooted, point-labeled trees with ordered branches) used in genetic programming and the labeled cyclic graphs germane to electrical circuits. The principles of developmental biology provide the motivation for mapping trees into circuits by means of a developmental process that begins with a simple embryo. For circuits, the initial circuit typically includes a test fixture consisting of certain fixed components (such as a source resistor, a load resistor, an input port, and an output port) as well as an embryo consisting of one or more modifiable wires. Until the modifiable wires are modified, the circuit does not produce interesting output. An electrical circuit is developed by progressively applying the functions in a circuit-constructing program tree to the modifiable wires of the embryo (and, during the developmental process, to succeeding modifiable wires and components). A single electrical circuit is created by executing the functions in an individual circuit-constructing program tree from the population. The functions are progressively applied in a developmental process to the embryo and its successors until all of the functions in the program tree are executed. That is, the functions in the circuit-constructing program tree progressively side-effect the embryo and its successors until a fully developed circuit eventually emerges. The functions are applied in a breadth-first order.

The functions in the circuit-constructing program trees are divided into five categories:

(1) topology-modifying functions that alter the topology of a developing circuit,

(2) component-creating functions that insert components into a developing circuit,

(3) development-controlling functions that control the development process by which the embryo and its successors become a fully developed circuit,

(4) arithmetic-performing functions that appear in subtrees as argument(s) to the component-creating functions and specify the numerical value of the component, and

(5) automatically defined functions that appear in the automatically defined functions and potentially enable certain substructures of the circuit to be reused (with parameterization).

3.1.1. Lowpass Filter Circuit

A simple *filter* is a one-input, one-output electronic circuit that receives a signal as its input and passes the frequency components of the incoming signal that lie in a specified range (called the *passband*) while suppressing the frequency components that lie in all other frequency ranges (the *stopband*).

A *lowpass filter* passes all frequencies below a certain specified frequency, but stops all higher frequencies.

Filter may be constructed from inductors and capacitors. The desired lowpass LC filter has a passband below 1,000 Hz and a stopband above 2,000 Hz. The circuit is driven by an incoming AC voltage source with a 2 volt amplitude. The *attenuation* of the filter is defined in terms of the output signal relative to the reference voltage (half of 2 volt here). In this problem, a voltage in the passband of exactly 1 volt and a voltage in the stopband of exactly 0 volts is regarded as ideal. The (preferably small) variation within the passband is called the *passband ripple*. Similarly, the incoming signal is never fully reduced to zero in the stopband of an actual filter. The (preferably small) variation within the stopband is called the *stopband ripple*. A voltage in the passband of between 970 millivolts and 1 volt (i.e., a passband ripple of 30 millivolts or less) and a voltage in the stopband of between 0 volts and 1 millivolts (i.e., a stopband ripple of 1 millivolts or less) is regarded as acceptable. Any voltage lower than 970 millivolts in the passband and any voltage above 1 millivolts in the stopband is regarded as unacceptable.

3.1.1.1. Preparatory steps for Lowpass Filter Circuit

Before applying genetic programming to a problem of circuit design, seven major preparatory steps are required: (1) identify the embryonic circuit, (2) determine the architecture of the circuit-constructing program trees, (3) identify the primitive functions of the program trees, (4) identify the terminals of the program trees, (5) create the fitness measure, (6) choose control parameters for the run, and (7) determine the termination criterion and method of result designation. A detailed discussion concerning how to apply these seven preparatory steps to a particular problem of circuit synthesis (such as a lowpass filter) is found in Koza, Bennett, Andre, and Keane 1999 (chapter 25).

The initial circuit used on a particular problem depends on the circuit's number of inputs and outputs. All development originates from the modifiable wires of the embryo within the initial circuit. An embryo with two modifiable wires (`Z0` and `Z1`) was used for the lowpass filter circuit

The architecture of each circuit-constructing program tree depends on the embryo. There is one result-producing branch in the program tree for each modifiable wire in the embryo.

The function set for each design problem depends on the type of electrical components that are to be used for constructing the circuit.

For the problem of synthesizing a lowpass filter, the function set included two component-creating functions (for inserting inductors and capacitors into a developing circuit), topology-modifying functions (for performing series and parallel divisions and for flipping components), one development-controlling function ("no operation"), functions for creating a via (direct connection) to ground, and functions for connecting pairs of points. That is, the function set, $\mathcal{F}_{\text{ccs-initial}}$, for each construction-continuing subtree was

$\mathcal{F}_{\text{ccs-initial}}$ = {`L`, `C`, `SERIES`, `PARALLEL0`, `PARALLEL1`, `FLIP`, `NOOP`, `T_GND_0`, `T_GND_1`, `PAIR_CONNECT_0`, `PAIR_CONNECT_1`}.

For additional details of these functions (and other aspects of this problem), see Koza, Bennett, Andre, and Keane 1999.

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

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

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

$\mathcal{T}_{\text{aps-initial}} = \{\Re\}$,

where $\Re$ represents floating-point random constants from –1.0 to +1.0.

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

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

The terminal and function sets were identical for all result-producing branches for a particular problem.

The evolutionary process is driven by the *fitness measure*. Each individual computer program in the population is executed and then evaluated, using the fitness measure. The nature of the fitness measure varies with the problem. The high-level statement of desired circuit behavior is translated into a well-defined measurable quantity that can be used by genetic programming to guide the evolutionary process.

The evaluation of each individual circuit-constructing program tree in the population begins with its execution. This execution progressively applies the functions in each program tree to the initial circuit, thereby creating a fully developed circuit. A netlist is created that identifies each component of the developed circuit, the nodes to which each component is connected, and the value of each component. The netlist becomes the input to our modified version of the 217,000-line SPICE (Simulation Program with Integrated Circuit Emphasis) simulation program(Quarles, Newton, Pederson, and Sangiovanni-Vincentelli 1994). SPICE then determines the behavior of the circuit.

Since the high-level statement of behavior for the desired circuit is expressed in terms of frequencies, the voltage VOUT is measured in the frequency domain. SPICE performs an AC small signal analysis and reports the circuit's behavior over five decades (between 1 Hz and 100,000 Hz) with each decade being divided into 20 parts (using a logarithmic scale), so that there are a total of 101 fitness cases.

Fitness is measured in terms of the sum over these cases of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit at the probe point VOUT and the target value for voltage. The smaller the value of fitness, the better. A fitness of zero represents an (unattainable) ideal filter. Specifically, the standardized fitness is

$$F(t) = \sum_{i=0}^{100} (W(d(f_i), f_i) d(f_i))$$

where f_i is the frequency of fitness case i; $d(x)$ is the absolute value of the difference between the target and observed values at frequency x; and $W(y,x)$ is the weighting for difference y at frequency x.

The fitness measure is designed to not penalize ideal values, to slightly penalize every acceptable deviation, and to heavily penalize every unacceptable deviation. Specifically, the procedure for each of the 61 points in the 3-decade interval between 1 Hz and 1,000 Hz for the intended passband is as follows:

- If the voltage equals the ideal value of 1.0 volt in this interval, the deviation is 0.0.
- If the voltage is between 970 millivolts and 1 volt, the absolute value of the deviation from 1 volt is weighted by a factor of 1.0.
- If the voltage is less than 970 millivolts, the absolute value of the deviation from 1 volt is weighted by a factor of 10.0.

The acceptable and unacceptable deviations for each of the 35 points from 2,000 Hz to 100,000 Hz in the intended stopband are similarly weighed (by 1.0 or 10.0) based on the amount of deviation from the ideal voltage of 0 volts and the acceptable deviation of 1 millivolts.

For each of the five "don't care" points between 1,000 and 2,000 Hz, the deviation is deemed to be zero.

Many of the random initial circuits and many that are created by the crossover and mutation operations in subsequent generations cannot be simulated by SPICE. These circuits receive a high penalty value of fitness (10^8) and become the worst-of-generation programs for each generation.

The population size was 640,000.

The problem was run on a medium-grained parallel Parsytec computer system consisting of 64 80-MHz PowerPC 601 processors arranged in an 8 by 8 toroidal mesh with a host PC Pentium type computer. The distributed genetic algorithm was used in which 64 demes (semi-isolated subpopulations) of 10,000 resided at each node of the parallel computer. 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 adjacent processing nodes.

3.1.1.2. Results for Lowpass Filter Circuit

Figure 1 shows the frequency domain behavior of an illustrative lowpass filter in which the boundary of the passband is at 1,000 Hz and the boundary of the stopband is 2,000 Hz. The horizontal axis represents the frequency of the incoming signal and ranges over five decades of frequencies between 1 Hz and 100,000 Hz on a logarithmic scale. The vertical axis represents the peak voltage of the output and ranges between 0 to 1 Volts on a linear scale. This figure shows that when the input to the circuit consists of a sinusoidal signal with any frequency from 1 Hz to 1,000 Hz, the output is a sinusoidal signal with an amplitude of a full 1 Volt. This figure also shows that when the input to the circuit consists of a sinusoidal signal with any frequency from 2,000 Hz to 100,000 Hz, the amplitude of the output is essentially 0 Volts. The region between 1,000 Hz and 2,000 Hz is a transition region where the voltage varies between 1 Volts (at 1,000 Hz) and essentially 0 Volts (at 2,000 Hz).

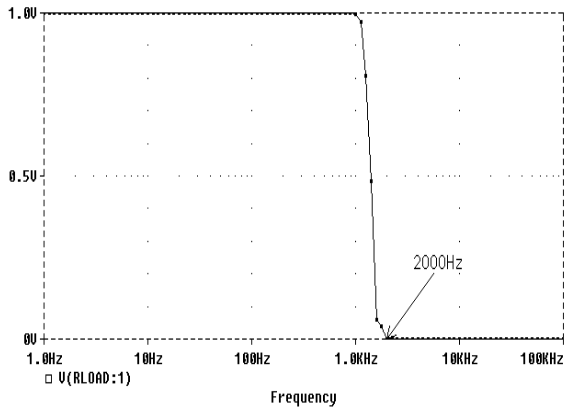

Figure 1 Frequency domain behavior of a lowpass filter.

Genetic programming is capable of automatically creating both the topology and sizing (component values) for lowpass filters (and other filter circuits, such as highpass filters, bandpass filters, bandstop filters, and filters with multiple passbands and stopbands).

A filter circuit may be evolved using a fitness measure based on frequency domain behavior. In particular, the fitness of an individual circuit is the sum, over 101 values of frequency between 1 Hz and 100,000 Hz (equally space on a logarithmic scale), of the absolute value of the difference between the individual circuit's output voltage and the ideal voltage for an ideal lowpass filter for that frequency (i.e., the voltages shown in figure 1).

For example, one run of genetic programming synthesized the lowpass filter circuit of figure 2.

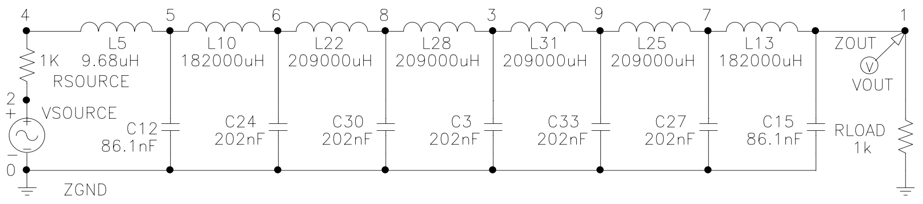

Figure 2 Lowpass filter created by genetic programming that infringes on Campbell's patent.

The evolved circuit of figure 2 is what is now called a cascade (ladder) of identical π sections (Koza, Bennett, Andre, and Keane 1999, chapter 25). The evolved circuit has the recognizable topology of the circuit for which George Campbell of American Telephone and Telegraph received U. S. patent 1,227,113 in 1917. Claim 2 of Campbell's patent covered,

> "An electric wave filter consisting of a connecting line of negligible attenuation composed of a plurality of sections, each section including a capacity element and an inductance element, one of said elements of each section being in series with the line and the other in shunt across the line, said capacity and inductance elements having precomputed values dependent upon the upper limiting frequency and the lower limiting frequency of a range of frequencies it is desired to transmit without attenuation, the values of said capacity and inductance elements being so proportioned that the structure transmits with practically negligible attenuation sinusoidal currents of all frequencies lying between said two limiting frequencies, while attenuating and approximately extinguishing currents of neighboring frequencies lying outside of said limiting frequencies."

In addition to possessing the topology of the Campbell filter, the numerical values of all the components in the evolved circuit closely approximate the numerical values taught in Campbell's 1917 patent.

Another run of genetic programming synthesized the lowpass filter circuit of figure 3. As before, this circuit was evolved using the previously described fitness measure based on frequency domain behavior.

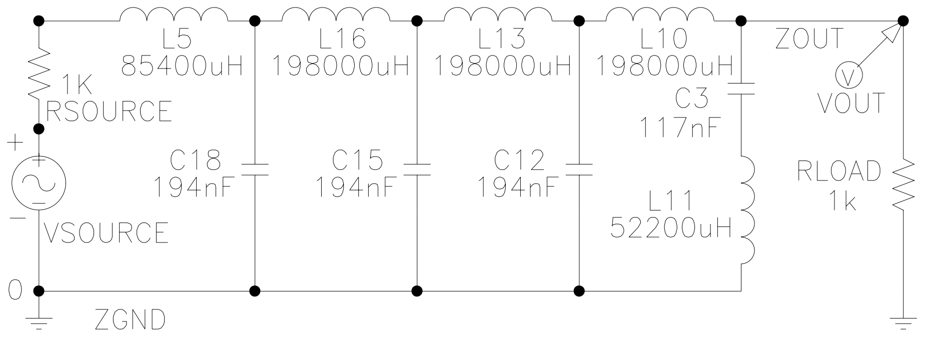

Figure 3 Lowpass filter created by genetic programming that infringes on Zobel's patent.

This evolved circuit differs from the Campbell filter in that its final section consists of both a capacitor and inductor. This filter is an improvement over the Campbell filter because its final section confers certain performance advantages on the circuit. This circuit is equivalent to what is called a cascade of three symmetric T-sections and an *M*-derived half section (Koza, Bennett, Andre, and Keane 1999, chapter 25). Otto Zobel of American Telephone and Telegraph Company invented and received a patent for an "*M*-derived half section" used in conjunction with one or more "constant K" sections. Again, the numerical values of all the components in this evolved circuit closely approximate the numerical values taught in Zobel's 1925 patent.

Seven circuits created using genetic programming infringe on previously issued patents ((Koza, Bennett, Andre, and Keane 1999). Others duplicate the functionality of previously patented inventions in novel ways.

In both of the foregoing examples, genetic programming automatically created both the topology and sizing (component values) of the entire filter circuit by using a fitness measure expressed in terms of the signal observed at the single output point (the probe point labeled VOUT in the figures).

3.1.2. Squaring Computational Circuit

An analog electrical circuit whose output is a well-known mathematical function (e.g., square, square root) is called a *computational circuit*.

We use a squaring computational circuit to illustrate the generality of the genetic programming process. Relatively few changes must be made in order to automatically evolve a squaring computational circuit.

First, computational circuits are typically constructed using capacitors, resistors, diodes, and transistors (instead of the inductors or capacitors used in the lowpass filter problem). Therefore, the function set must contain functions capable of inserting these components into a developing circuit.

Second, since the purpose of the circuit is different, the fitness measure must reflect the fact that the circuit's output is intended to be a voltage equal to the square of the circuit's input.

3.1.2.1. Preparatory Steps for Squaring Computational Circuit

For the problem of synthesizing a computational circuit, capacitors, resistors, diodes, and transistors were used (instead of the inductors or capacitors used in the lowpass filter problem). The function set also included functions to provide connectivity to the positive and negative power supplies (in order to provide a source of energy for the transistors). Thus, the function set, $\mathcal{F}_{\text{ccs-initial}}$, for each construction-continuing subtree was

$\mathcal{F}_{\text{ccs-initial}}$ = {R, C, SERIES, PARALLEL0, PARALLEL1, FLIP, NOOP, T_GND_0, T_GND_1, T_POS_0, T_POS_1, T_NEG_0, 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.

Because filters discriminate on incoming signals based on frequency, the lowpass filter circuit was automatically synthesized using a fitness measure based on the behavior of the circuit in the frequency domain. For the squaring computational circuit, SPICE is called to perform a DC sweep analysis at 21 equidistant voltages between –250 millivolts and +250 millivolts. Fitness is the sum, over these 21 fitness cases, of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit and the target value for voltage (i.e., the square of the input voltage).

3.1.2.2. Results for Squaring Computational Circuit

Figure 4 shows a squaring circuit composed of transistors, capacitors, and resistors that was automatically synthesized using a fitness measure based on the behavior of the circuit in the time domain (Mydlowec and Koza 2000).

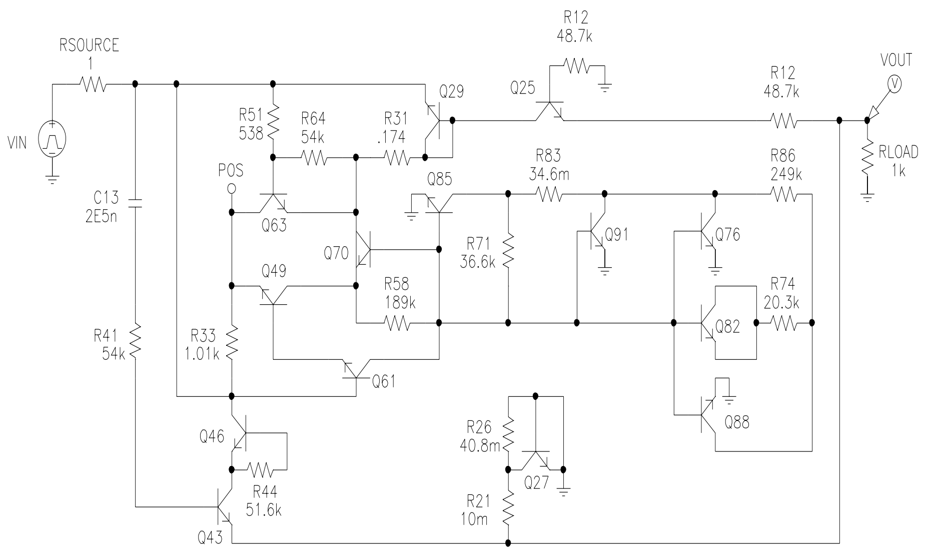

Figure 4 Squaring circuit created by genetic programming.

This circuit was evolved using a fitness measure based on time-varying input signals. In particular, fitness was the sum, taken at certain sampled times for four different time-varying input signals, of the absolute value of the difference between the individual circuit's output voltage and the desired output voltage (i.e., the square of the voltage of the input signal at the particular sampled time).

The four input signals were structured to provide a representative mixture of input values. All of the input signals produce outputs that are well within the range of voltages that can be handled by transistors (i.e., below 4 volts). For example, one of the input signals is a rising ramp whose value remains at 0 up to 0.2 seconds and then rises to 2 Volts between 0.2 seconds and 1.0 seconds. Figure 5 shows the output voltage produced by the evolved circuit for the rising ramp input superimposed on the (virtually indistinguishable) correct output voltage for the squaring function. As can be seen, as soon as the input signal becomes non-zero, the output is a parabolic-shaped curve representing the square of the incoming voltage.

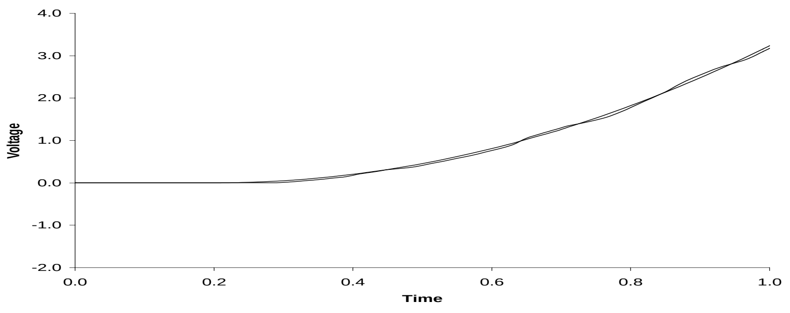

Figure 5 Output for rising ramp input for squaring circuit.

4. Automatic Synthesis of Controllers

Genetic programming is capable of automatically creating both the topology and sizing (tuning) for controllers composed of time-domain blocks (such as integrators, differentiators, multipliers, adders, delays, gains, leads, and lags) merely by specifying the controller's effect on the to-be-controlled plant (Keane, Yu, and Koza 2000; Koza, Keane, Yu, Bennett, Mydlowec, and Stiffelman 1999; Koza, Keane, Bennett, Yu, Mydlowec, and Stiffelman, Oscar 1999; Koza, Keane, Yu, Bennett, and Mydlowec 2000; Koza, Keane, Yu, Mydlowec, and Bennett 2000a, 2000b; Koza, Yu, Keane, and Mydlowec 2000; Yu, Keane, and Koza. 2000). This automatic synthesis of controllers from data is performed by genetic programming even though there is no general mathematical method for creating both the topology and sizing for controllers from a high-level statement of the design goals for the controller.

In the PID type of controller, the controller's output is the sum of proportional (P), integrative (I), and derivative (D) terms based on the difference between the plant's output and the reference signal. Albert Callender and Allan Stevenson of Imperial Chemical Limited of Northwich, England received U.S. Patent 2,175,985 in 1939 for the PI and PID controller.

Claim 1 of Callender and Stevenson (1939) covers what is now called the PI controller,

> "A system for the automatic control of a variable characteristic comprising means proportionally responsive to deviations of the characteristic from a desired value, compensating means for adjusting the value of the characteristic, and electrical means associated with and actuated by responsive variations in said responsive means, for operating the compensating means to correct such deviations in conformity with the sum of the extent of the deviation and the summation of the deviation."

Claim 3 of Callender and Stevenson (1939) covers what is now called the PID controller,

> "A system as set forth in claim 1 in which said operation is additionally controlled in conformity with the rate of such deviation."

The vast majority of automatic controllers used by industry are of the PI or PID type. However, it is generally recognized by leading practitioners in the field of control that PI and PID controllers are not ideal (Astrom and Hagglund 1995; Boyd and Barratt 1991).

There is no preexisting general-purpose analytic method (prior to genetic programming) for automatically creating both the topology and tuning of a controller for arbitrary linear and non-linear plants that can simultaneously optimize prespecified performance metrics. The performance metrics used in the field of control include, among others,

- minimizing the time required to bring the plant output to the desired value (as measured by, say, the integral of the time-weighted absolute error),
- satisfying time-domain constraints (involving, say, overshoot and disturbance rejection),
- satisfying frequency domain constraints (e.g., bandwidth), and
- satisfying additional constraints, such as limiting the magnitude of the control variable or the plant's internal state variables.

We employ a problem involving control of a two-lag plant (described by Dorf and Bishop 1998, page 707) to illustrate the automatic synthesis of controllers by means of genetic programming. The problem entails synthesizing the design of both the topology and parameter values for a controller for a two-lag plant such that plant output reaches the level of the reference signal so as to minimize the integral of the time-weighted absolute error, such that the overshoot in response to a step input is less than 2%, and such that the controller is robust in the face of significant variation in the plant's internal gain, K, and the plant's time constant, τ.

Genetic programming routinely creates PI and PID controllers infringing on the 1942 of Callender and Stevenson patent during intermediate generations of runs of genetic programming on controller problems. However, the PID controller is not the best possible controller for this (and many) problems.

Figure 6 shows the block diagram for the best-of-run controller evolved during one run of this problem. In this figure, $R(s)$ is the reference signal; $Y(s)$ is the plant output; and $U(s)$ is the controller's output (control variable). This evolved controller is 2.42 times better than the Dorf and Bishop (1998) controller as measured by the criterion used by Dorf and Bishop. In addition, this evolved controller has only 56% of the rise time in response to the reference input, has only 32% of the settling time, and is 8.97 times better in terms of suppressing the effects of disturbance at the plant input.

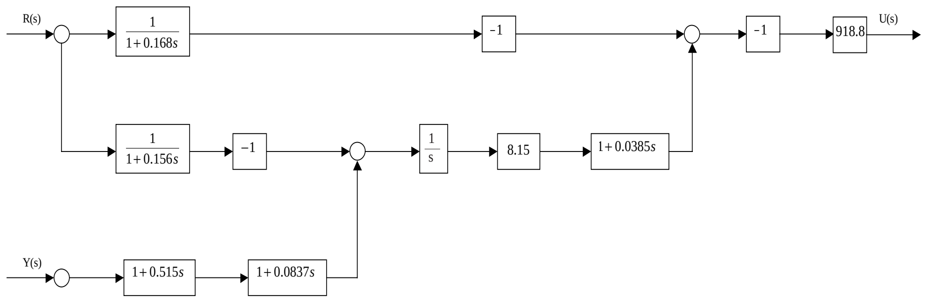

Figure 6 Evolved controller that infringes on Jones' patent.

This genetically evolved controller differs from a conventional PID controller in that it employs a second derivative processing block. Specifically, after applying standard manipulations to the block diagram of this evolved controller, the transfer function for the best-of-run controller can be expressed as a transfer function for a pre-filter and a transfer function for a compensator. The transfer function for the pre-filter, $G_{p32}(s)$, for the best-of-run individual from generation 32 is

$$G_{p32}(s) = \frac{1(1+.1262s)(1+.2029s)}{(1+.03851s)(1+.05146)(1+.08375)(1+.1561s)(1+.1680s)}$$

The transfer function for the compensator, $G_{c32}(s)$, is

$$G_{c32}(s) = \frac{7487(1+.03851s)(1+.05146s)(1+.08375s)}{s} = \frac{7487.05+1300.63s+71.2511s^2+1.2426s^3}{s}$$

The s^3 term (in conjunction with the s in the denominator) indicates a second derivative. Thus, the compensator consists of a second derivative in addition to proportional, integrative, and derivative functions. As it happens, Harry Jones of The Brown Instrument Company of Philadelphia received U. S. Patent 2,282,726 for this kind of controller topology in 1942.

Claim 38 of the Jones patent (Jones 1942) states,

> "In a control system, an electrical network, means to adjust said network in response to changes in a variable condition to be controlled, control means responsive to network adjustments to control said condition, reset means including a reactance in said network adapted following an adjustment of said network by said first means to initiate an additional network adjustment in the same sense, and rate control means included in said network adapted to control the effect of the first mentioned adjustment in accordance with the second or higher derivative of the magnitude of the condition with respect to time."

Note that the human user of genetic programming did not preordain, prior to the run (i.e., as part of the preparatory steps for genetic programming), that a second derivative should be used in the controller (or, from that matter, even that a P, I, or D block should be used). Genetic programming automatically discovered that the second derivative element (along with the P, I, and D elements) were useful in producing a good controller for this particular problem. That is, necessity was the mother of invention.

Similarly, the human who initiated this run of genetic programming did not preordain any particular topological arrangement of proportional, integrative, derivative, second derivative, or other functions within the automatically created controller. Instead, genetic programming automatically created a controller for the given plant without the benefit of user-supplied information concerning the total number of processing blocks to be employed in the controller, the type of each processing block, the topological interconnections between the blocks, the values of parameters for the blocks, or the existence of internal feedback (none in this instance) within the controller.

5. Other Examples

In the same vein as the foregoing examples, it should be mentioned that genetic programming is capable of discovering both the topological and numerical aspects of a satisfactory antenna design from a high-level specification of the antenna's behavior. In one particular problem (Comisky, Yu, and Koza 2000), genetic programming automatically discovered the design for a satisfactory antenna composed of wires for maximizing gain in a preferred direction over a specified range of frequencies, having a reasonable value of voltage standing wave ratio when the antenna is fed by a transmission line with a specified characteristic impedance, and fitting into a specified bounding rectangle. The design that genetic programming discovered included

(1) the number of directors in the antenna,

(2) the number of reflectors,

(3) the fact that the driven element, the directors, and the reflector are all single straight wires,

(4) the fact that the driven element, the directors, and the reflector are all arranged in parallel,

(5) the fact that the energy source (via the transmission line) is connected only to the driven element — that is, the directors and reflectors are parasitically coupled.

The last three of the above characteristics discovered by genetic programming are the defining characteristics of an inventive design conceived in the early years of the field of antenna design (Uda 1926, 1927; Yagi 1928).

Similarly, genetic programming is capable of discovering both the topology of the network of chemical reactions and the numerical parameters of all the reactions of the network starting with observed time-domain concentrations of final product substance(s). The approach used (Koza, Mydlowec, Lanza, Yu, and Keane 2000) involved

(1) establishing a representation for chemical networks involving symbolic expressions (S-expressions) and program trees that are composed of functions and terminals and that can be progressively bred (and improved) by genetic programming,

(2) converting each individual program tree in the population into an analog electrical circuit representing a network of chemical reactions,

(3) obtaining the behavior of the individual network of chemical reactions by simulating the corresponding electrical circuit,

(4) defining a fitness measure that measures how well the behavior of an individual network matches the observed time-domain data concerning concentrations of product substances, and

(5) using the fitness measure to enable genetic programming to breed a population of improving program trees.

Figure 7 shows a network of chemical reactions that was evolved in this manner using genetic programming. This network closely matches the observed data for all data points, has the same topology as the correct metabolic pathway, and has the almost exactly the same rate constants for the four reactions.

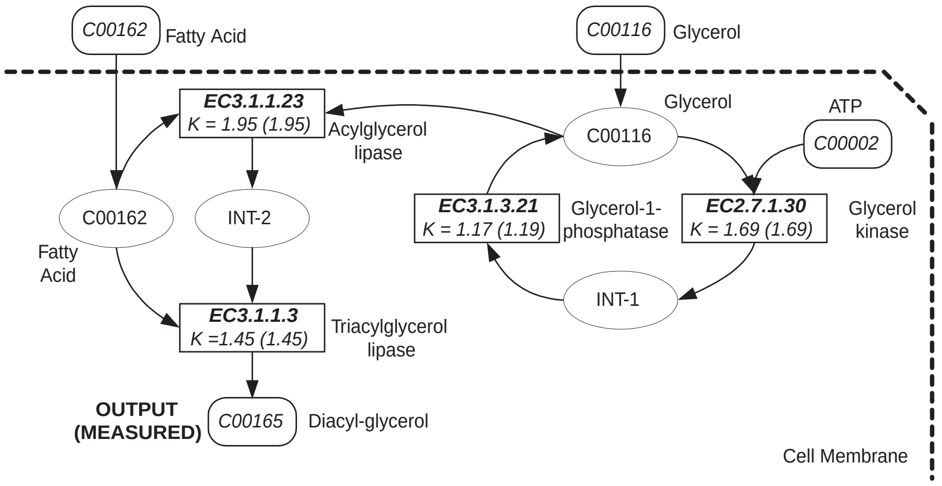

Figure 7 Network of chemical reactions that was evolved using genetic programming.

This network was evolved using only by the time-domain concentration values of the final product `C00165` (diacyl-glycerol). genetic programming created the entire metabolic pathway, including

- topological features such as the internal feedback loop,
- topological features such as a bifurcation point where one substance is distributed to two different reactions,
- topological features such as an accumulation point where one substance is accumulated from two sources,
- use of two intermediate substances (`INT_1` and `INT_2`)
- numerical rates (sizing) for all reactions.

6. Conclusions

This chapter has demonstrated that a biologically motivated algorithm (genetic programming) is capable of automatically synthesizing the design of both the topology of complex graphical structures and optimal or near-optimal numerical values for all elements of the structure possessing parameters.

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.

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

Astrom, Karl J. and Hagglund, Tore. 1995. *PID Controllers: Theory, Design, and Tuning.* Second Edition. Research Triangle Park, NC: Instrument Society of America.

Banzhaf, Wolfgang, Daida, Jason, Eiben, A. E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E. (editors). 1999. *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 13-17, 1999, Orlando, Florida USA*. San Francisco, CA: Morgan Kaufmann.

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. 1998. *Genetic Programming – An Introduction*. San Francisco, CA: Morgan Kaufmann and Heidelberg: dpunkt.

Banzhaf, Wolfgang, Poli, Riccardo, Schoenauer, Marc, and Fogarty, Terence C. 1998. *Genetic Programming: First European Workshop. EuroGP'98. Paris, France, April 1998 Proceedings. Paris, France. April l998*. Lecture Notes in Computer Science. Volume 1391. Berlin, Germany: Springer-Verlag.

Boyd, S. P. and Barratt, C. H. 1991. *Linear Controller Design: Limits of Performance*. Englewood Cliffs, NJ: Prentice Hall.

Callender, Albert and Stevenson, Allan Brown. 1939. *Automatic Control of Variable Physical Characteristics*. United States Patent 2,175,985. Filed February 17, 1936 in United States. Filed February 13, 1935 in Great Britain. Issued October 10, 1939 in United States.

Campbell, George A. 1917. *Electric Wave Filter*. Filed July 15, 1915. U. S. Patent 1,227,113. Issued May 22, 1917.

Comisky, William, Yu, Jessen, and Koza, John. 2000. Automatic synthesis of a wire antenna using genetic programming. *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Las Vegas, Nevada*. Pages 179 - 186.

Dorf, Richard C. and Bishop, Robert H. 1998. *Modern Control Systems*. Eighth edition. Menlo Park, CA: Addison-Wesley.

Holland, John H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press 1975. Second edition. Cambridge, MA: The MIT Press 1992.

Jones, Harry S. 1942. *Control Apparatus*. United States Patent 2,282,726. Filed October 25, 1939. Issued May 12, 1942.

Keane, Martin A., Yu, Jessen, and Koza, John R. 2000. Automatic synthesis of both the topology and tuning of a common parameterized controller for two families of plants using genetic programming. In Whitley, Darrell, Goldberg, David, Cantu-Paz, Erick, Spector, Lee, Parmee, Ian, and Beyer, Hans-Georg (editors). *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, July 10 - 12, 2000, Las Vegas, Nevada*. San Francisco: Morgan Kaufmann Publishers. Pages 496 - 504. 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., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick. (editors). 1998. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. San Francisco, CA: Morgan Kaufmann.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999. *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A., and Brave Scott. 1999. *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann.

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

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., Keane, Martin A., Bennett, Forrest H III, Yu, Jessen, Mydlowec, William, and Stiffelman, Oscar. 1999. Automatic creation of both the topology and parameters for a robust controller by means of genetic programming. *Proceedings of the 1999 IEEE International Symposium on Intelligent Control, Intelligent Systems, and Semiotics*. Piscataway, NJ: IEEE. Pages 344 - 352.

Koza, John R., Keane, Martin A., Yu, Jessen, Bennett, Forrest H III, and Mydlowec, William. 2000. Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*. (1) 121 - 164.

Koza, John R., Keane, Martin A., Yu, Jessen, Bennett, Forrest H III, Mydlowec, William, and Stiffelman, Oscar. 1999. Automatic synthesis of both the topology and parameters for a robust controller for a non-minimal phase plant and a three-lag plant by means of genetic programming. *Proceedings of 1999 IEEE Conference on Decision and Control*. Pages 5292 - 5300.

Koza, John R., Keane, Martin A., Yu, Jessen, Mydlowec, William, and Bennett, Forrest H III. 2000a. Automatic synthesis of both the topology and parameters for a controller for a three-lag plant with a five-second delay using genetic programming. In Cagnoni, Stafano, Poli, Riccardo, Smith, George D., Corner, David, Oates, Martin, and Hart, Emma (editors). *Real-World Applications of Evolutionary Computing. EvoWorkshops 2000. EvoIASP, Evo SCONDI, EvoTel, EvoSTIM, EvoRob, and EvoFlight, Edinburgh, Scotland, UK, April 2000, Proceedings*. Lecture Notes in Computer Science. Volume 1803. Berlin, Germany: Springer-Verlag. Pages 168 - 177. ISBN 3-540-67353-9.

Koza, John R., Keane, Martin A., Yu, Jessen, Mydlowec, William, and Bennett, Forrest H III. 2000b. Automatic synthesis of both the control law and parameters for a controller for a three-lag plant with five-second delay using genetic programming and simulation techniques. In *Proceedings of the 2000 American Control Conference, Chicago, Illinois, June 28 - 30, 2000*. Evanston, IL: American Automatic Control Council. Pages 453-459.

Koza, John R., Mydlowec, William, Lanza, Guido, Yu, Jessen, and Keane, Martin A. 2000. *Reverse Engineering and Automatic Synthesis of Metabolic Pathways from Observed Data Using Genetic Programming*. Stanford Medical Informatics Technical Report SMI-2000-0851.

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

Koza, John R., Yu, Jessen, Keane, Martin A., and Mydlowec, William. 2000. Evolution of a controller with a free variable using genetic programming. In Poli, Riccardo, Banzhaf, Wolfgang, Langdon, William B., Miller, Julian, Nordin, Peter, and Fogarty, Terence C. 2000. *Genetic Programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 2000, Proceedings*. Lecture Notes in Computer Science. Volume 1802. Berlin, Germany: Springer-Verlag. Pages 91 - 105. ISBN 3-540-67339-3.

Langdon, William B. 1998. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Amsterdam: Kluwer.

Mydlowec, William and Koza, John. 2000. Use of time-domain simulations in automatic synthesis of computational circuits using genetic programming. *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Las Vegas, Nevada*. Pages 187 - 197.

Poli, Riccardo, Nordin, Peter, Langdon, William B., and Fogarty, Terence C. 1999. *Genetic Programming: Second European Workshop. EuroGP'99. Proceedings*. Lecture Notes in Computer Science. Volume 1598. Berlin, Germany: Springer-Verlag.

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.

Sterling, Thomas L., Salmon, John, and Becker, Donald J., and Savarese, Daniel F. 1999. *How to Build a Beowulf: A Guide to Implementation and Application of PC Clusters*. Cambridge, MA: MIT Press.

Uda, S. 1926. Wireless beam of short electric waves. *Journal of the IEE (Japan)*. March 1926. 273 - 282.

Uda, S. 1927. Wireless beam of short electric waves. *Journal of the IEE (Japan)*. March 1927.1209-1219.

Whitley, Darrell, Goldberg, David, Cantu-Paz, Erick, Spector, Lee, Parmee, Ian, and Beyer, Hans-Georg (editors). 2000. *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, July 10 - 12, 2000, Las Vegas, Nevada*. San Francisco: Morgan Kaufmann Publishers.

Wong, Man Leung and Leung, Kwong Sak. 2000. *Data Mining Using Grammar Based Genetic Programming and Applications*. Amsterdam: Kluwer Academic Publishers. ISBN: 0-7923-7746-X

Yagi, H. 1928. Beam transmission of ultra short waves. *Proceedings of the IRE*. 26: 714-741. June 1928.

Yu, Jessen, Keane, Martin A., and Koza, John R. 2000. Automatic design of both topology and tuning of a common parameterized controller for two families of plants using genetic programming. In *Proceedings of Eleventh IEEE International Symposium on Computer-Aided Control System Design (CACSD) Conference and Ninth IEEE International Conference on Control Applications (CCA) Conference, Anchorage, Alaska, September 25 - 27, 2000*. Pages CACSD-234 - 242.

Zobel, Otto Julius. 1925. *Wave Filter*. Filed January 15, 1921. U. S. Patent 1,538,964. Issued May 26, 1925.