Genetic Programming as a Darwinian Invention Machine

John R. Koza
Section on Medical Informatics
Department of Medicine
Stanford, California 94305
`koza@stanford.edu`
`http://www.smi.stanford.edu/people/koza/`

Forrest H Bennett III
Genetic Programming Inc.
Los Altos, California 94023
`forrest@evolute.com`

Oscar Stiffelman
Computer Science Department
Stanford University
Stanford, California 94305
`ozzie@cs.stanford.edu`

ABSTRACT

Genetic programming is known to be capable of creating designs that satisfy prespecified high-level design requirements for analog electrical circuits and other complex structures. However, in the real world, it is often important that a design satisfy various non-technical requirements. One such requirement is that a design not possess the key characteristics of any previously known design. This paper shows that genetic programming can be used to generate novel solutions to a design problem so that genetic programming may be potentially used as an invention machine. This paper turns the clock back to the period just before the time (1917) when George Campbell of American Telephone and Telegraph invented and patented the design for an electrical circuit that is now known as the ladder filter. Genetic programming is used to reinvent the Campbell filter. The paper then turns the clock back to the period just before the time (1928) when Wilhelm Cauer invented and patented the elliptic filter. Genetic programming is then used to reinvent a technically equivalent filter that avoids the key characteristics of then-preexisting Campbell filter. Genetic programming can be used as an invention machine by employing a two-part fitness measure that incorporates both the degree to which an individual in the population satisfies the given technical requirements and the degree to which the individual does not possess the key characteristics of preexisting technology.

1 Introduction

Design is a major activity of practicing engineers. The design process entails the creation (synthesis) of a complex structure to satisfy user-defined requirements. 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. Consequently, the field of design is a source of challenging problems for automated techniques of machine intelligence. In particular, design problems can test whether an automated technique can produce results that are competitive with human-produced results.

Design requirements in the real world often include important non-technical considerations. For example, a practicing engineer will often be asked to create a design that does not possess the key characteristics of any previously known solution to the problem at hand. Novelty may be desirable for several reasons. Novelty enables a company to obtain patent protection for its product, to avoid infringement ("engineer around") a preexisting patent (often a competitor's patent), or simply to differentiate its product in the marketplace on the basis of its unique technology. Regardless of the motivation, avoidance of "prior art" (the term used in the patent law to describe a field's preexisting technology, whether patented or not) is often important in the real world. Avoidance of prior art is accomplished by creating a design that does not "read on" (i.e., possess the key characteristics of) the prior art. In order to be patentable, a designed entity must be both "useful" and "new."

Genetic programming is an automatic technique that is capable of creating designs. Genetic programming approaches a design problem in terms of "what needs to be done" as opposed to "how to do it". For example, genetic programming has demonstrated that it is capable of synthesizing the design of a wide variety of analog electrical circuits and other complex structures (Koza, Bennett, Andre, and Keane 1999). Genetic programming often creates novel designs because it employs a probabilistic process to evolve designs and because it is not encumbered by the preconceptions that often channel human thinking down familiar paths. Although genetic programming has demonstrated an ability to automatically create useful entities (i.e., those that satisfy technical design requirements) and although it sometimes creates novel designs, none of the previously reported efforts have addressed the issue of actively avoiding the creation of an entity that reads on prior art.

This paper demonstrates that genetic programming can be modified to automatically create designs that satisfactorily solve a given problem while simultaneously avoiding prior art. This is accomplished using an illustrative problem involving the synthesis of the design of a lowpass filter circuit. Section 2 reviews how genetic programming has been successfully applied to the problem of synthesizing the topology and sizing of electrical circuits. Section 3 states the illustrative problem. Section 4 presents the preparatory steps necessary for applying our method of synthesizing novel designs to the illustrative problem. Section 5 presents the results.

2 Automatic Creation of Circuit Topology and Sizing

Genetic programming (Koza 1992; Koza and Rice 1992) is an extension of the genetic algorithm (Holland 1975) that automatically creates computer programs to solve problems. Genetic programming is also capable of evolving multi-part programs (Koza 1994a, 1994b) consisting of a main program and one or more reusable, parametrized, hierarchically-callable automatically defined functions (subroutines). Architecture-altering operations (Koza 1995) enable genetic programming to automatically determine the number of subroutines, the number of arguments that each possesses, and the nature of the hierarchical references, if any, among the subroutines. Architecture-altering operations also enable genetic programming to automatically determine whether and how to use internal memory, iterations, and recursion in evolved programs (Koza, Bennett, Andre, and Keane 1999). Additional information on current research in genetic programming can be found in Banzhaf, Nordin, Keller, and Francone 1998; Langdon 1998; Kinnear 1994; Angeline and Kinnear 1996; Spector, Langdon, O'Reilly, and Angeline 1999; 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; and Banzhaf, Poli, Schoenauer, and Fogarty 1998.

The design process for electrical circuits begins with a high-level description of the circuit's desired behavior and characteristics and entails creation of the topology and sizing of a satisfactory circuit. The *topology* of a circuit includes specifying the gross number of components in the circuit, the type of each component (e.g., a capacitor), and a *netlist* specifying where each lead of each component is to be connected. *Sizing* involves specifying the values (typically numerical) of each of the circuit's components. Until recently, there has previously been no general technique for automatically creating the topology and sizing for an analog electrical circuit from a high-level statement of the circuit's desired behavior and characteristics. In describing the design process for analog circuits, Aaserud and Nielsen (1995) observed,

> 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.

Genetic programming is capable of automatically creating (synthesizing) the design of both the circuit’s topology and sizing from a high-level statement of a circuit's desired behavior and characteristics (Koza, Bennett, Andre, Keane, and Dunlap 1997; Koza, Bennett, Andre, and Keane 1999). The evolved circuits include lowpass, highpass, bandpass, bandstop, crossover, multiple bandpass, and asymmetric bandpass filters, amplifiers, computational circuits, a temperature-sensing circuit, a voltage reference circuit, frequency-measuring circuits, robot controller, and source identification circuits. The evolved circuits include 11 previously patented circuits.

Genetic programming can be applied to circuit design by establishing a mapping between the rooted, point-labeled trees (i.e., acyclic graphs) with ordered branches used in genetic programming and the specialized type of cyclic graphs germane to electrical circuits. The creative work of Kitano (1990) on using developmental genetic algorithms to evolve neural networks, the innovative work of Gruau (1992, 1994a, 1994b) on using genetic programming to evolve neural networks (cellular encoding), and the principles of developmental biology suggest a method for mapping trees into electrical circuits by means of a growth process that begins with a simple embryo. For electrical circuits, the embryo includes one or more modifiable wires. The embryo is embedded into a test fixture consisting of certain fixed wires that provide connectivity to the circuit's external inputs and outputs and certain fixed (hard-wired) components (such as a source resistor and a load resistor). Until the modifiable wires are modified by the developmental process, the initial circuit (consisting of an embryo embedded into a text fixture) produces only trivial output. An electrical circuit is developed by progressively applying the functions in a circuit-constructing program tree (in the population being bred by genetic programming) to the modifiable wires of the original embryo and, during the developmental process, to newly created modifiable components and modifiable wires.

The functions in the circuit-constructing program trees are divided into five categories: (1) topology-modifying functions that alter the circuit topology, (2) component-creating functions that insert components into the circuit, (3) development-controlling functions that control the development process by which the embryo and its successors is changed into 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 enable certain substructures of the circuit to be reused (with parameterization).

An electrical circuit is created by executing the functions in a circuit-constructing program tree. The functions are progressively applied (in a breadth-first order) 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. Each branch of the program tree is created in accordance with a constrained syntactic structure. Each branch is composed of topology-modifying functions, component-creating functions, development-controlling functions, and terminals. Component-creating functions typically have one arithmetic-performing subtree, while topology-modifying functions, and development-controlling functions do not. Component-creating functions and topology-modifying functions are internal points of their branches and possess one or more arguments (construction-continuing subtrees) that continue the developmental process.

Each non-numeric function is associated with a modifiable wire or modifiable component in the developing circuit and modifies it in a specified manner. The construction-continuing subtree (if any) of a function points to a successor function or terminal in the circuit-constructing program tree. The arithmetic-performing subtree of a component-creating functions consists of a composition of arithmetic functions (addition and subtraction) and random constants (in the range -1.0 to +1.0). The arithmetic-performing subtree specifies the numerical value of a component by returning a floating-point value that is interpreted on a logarithmic scale in a range of 10 orders of magnitude (using a unit of measure appropriate for the particular type of component).

3 Statement of the Illustrative Problem

The method will be illustrated on the problem of creating the topology and sizing for a lowpass filter circuit. A simple *filter* is a one-input, one-output circuit that receives a signal 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*) (Williams and Taylor 1995). The desired lowpass filter is to pass all frequencies below 1,000 Hertz (Hz) and suppress all frequencies above 2,000 Hz. The circuit is driven by an incoming AC voltage source with a 2 volt amplitude. The circuit is tested by a test fixture containing a 1,000 Ohm (Ω) source (internal) resistor RSOURCE and a 1,000 Ω load resistor RLOAD. There should be a sharp drop-off from 1 Volt to 0 Volts in the transitional ("don't care") region between 1,000 Hz and 2,000 Hz.

A passband voltage of exactly 1 volt and a stopband voltage of exactly 0 volts is regarded as ideal. 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) are regarded as acceptable. A voltage lower than 970 millivolts in the passband or above 1 millivolts in the stopband is regarded as unacceptable.

The above design goals can be satisifed by many different circuits. Figure 1 shows a 100%-compliant circuit that was evolved using genetic programming (described in chapter 25 of Koza, Bennett, Andre, and Keane 1999). This evolved circuit consists of seven inductors (L5, L10, L22, L28, L31, L25, and L13) arranged horizontally across the top of the figure "in series" with the incoming signal VSOURCE and the source resistor RSOURCE. It also contains seven capacitors (C12, C24, C30, C3, C33, C27, and C15) that are each shunted to ground.

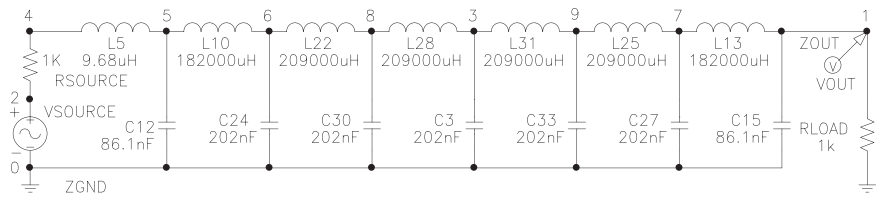

Figure 1 Seven-rung ladder lowpass filter.

This circuit has the recognizable features of the circuit for which George Campbell of American Telephone and Telegraph received U. S. patent 1,227,113 (Campbell 1917). Claim 2 of the 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.

An examination of the evolved circuit of figure 1 shows that it indeed consists of "a plurality of sections." (specifically, seven). In the figure, "Each section include[es] a capacity element and an inductance element." Specifically, the first of the seven sections consists of inductor L5 and capacitor C12; the second section consists of inductor L10 and capacitor C24; and so forth. Moreover, "one of said elements of each section [is] in series with the line and the other in shunt across the line." Inductor L5 of the first section is indeed "in series with the line" and capacitor C12 is indeed "in shunt across the line." This is also true for the circuit's remaining six sections. Moreover, figure 1 here matches figure 7 of Campbell's 1917 patent. In addition, this circuit's 100% compliant frequency domain behavior confirms the fact that the values of the inductors and capacitors are such as to transmit "with practically negligible attenuation sinusoidal currents" of the passband frequencies "while attenuating and approximately extinguishing currents" of the stopband frequencies. Thus, the evolved circuit reads on claim 2 of Campbell's 1917 patent. If this patent had not long since expired, the evolved circuit would infringe on the patent.

4 Preparatory Steps

Before applying genetic programming to a problem of circuit design, seven major preparatory steps are required: (1) identify the initial circuit (test fixture and embryo) of the developmental process, (2) determine the architecture of the circuit-constructing program trees, (3) identify the primitive functions, (4) identify the terminals, (5) create the fitness measure, (6) choose control parameters, and (7) determine the termination criterion and method of result designation.

4.1 Initial Circuit

An electrical circuit is created in a developmental process by executing a circuit-constructing program tree that contains various component-creating, topology-modifying, and development-controlling functions. An initial circuit consisting of an embryo and a test fixture is the starting point of a developmental process that transforms a program tree in the population into a fully developed electrical circuit. The embryo contains at least one modifiable wire. The test fixture is a fixed (hard-wired) substructure composed of nonmodifiable wires and nonmodifiable electrical components. The test fixture provides access to the circuit's external input(s) and permits probing of the circuit's output. An embryo is embedded into the test fixture. All development occurs in the embryo.

Figure 2 shows a one-input, one-output initial circuit consisting of an embryo embedded in a test fixture. The embryo consists of two modifiable wires Z0 and Z1. The test fixture consists of an incoming signal (input to the overall circuit) in the form of voltage source VSOURCE (2 Volt peak alternating-current) between nodes 0 and 1, a source resistor RSOURCE (whose value is 1,000 Ω) between nodes 1 and 2, a nonmodifiable wire ZOUT between nodes 3 and 5, a voltage probe point VOUT (the output of the overall circuit) at node 5, a load resistor RLOAD (whose value is 1,000 Ω) between nodes 5 and 0, and a nonmodifiable wire ZGND between nodes 4 and 0.

4.2 Program Architecture

Since there is one result-producing branch in the program tree for each modifiable wire in the embryo, the architecture of each circuit-constructing program tree has two result-producing branches.

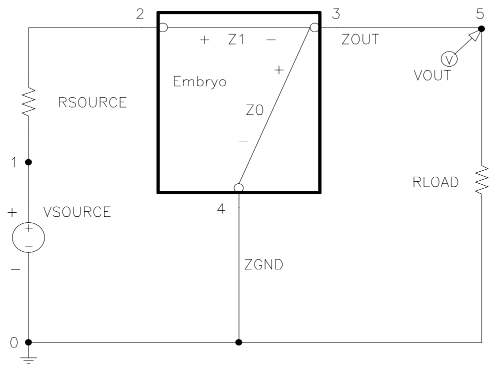

Figure 2 One-input, one-output initial circuit with two modifiable wires. **Function Set**

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

$\mathcal{F}_{\text{ccs}}$ = {`C`, `L`, `SERIES`, `PARALLEL0`, `FLIP`, `TVIA0`, …, `TVIA7`, `NOOP`}

All functions in this section are described in detail in Koza, Bennett, Andre, and Keane 1999. Briefly, the `C` and `L` functions are component-creating functions that insert an inductor or capacitor (respectively) into a developing circuit and that assign a numerical value to the inserted component. The `SERIES` and `PARALLEL0` functions modify the topology of the developing circuit by performing a series or parallel (respectively) division. The `FLIP` function reverse the polarity of a component. The eight `VIA` functions provides direct connectivity between two points within the developing circuit via one of eight numbered layers on the imaginary printed circuit board or piece of silicon on which the circuit resides. The `NOOP` (No operation) function is a development-controlling function.

4.4 Terminal Set

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

$\mathcal{T}_{\text{ccs}}$ = {`END`, `CUT`}.

Briefly, the development-controlling `END` function makes the modifiable wire or modifiable component with which it is associated non-modifiable (thereby ending a particular developmental path). The `CUT` function causes the highlighted component to be removed from the circuit.

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

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

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 is,

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

4.5 Fitness Measure

The evaluation of each individual circuit-constructing program tree in the population begins with its execution. The execution progressively applies the functions in the program tree to the embryo of the 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. SPICE is instructed to perform an AC small signal analysis and report the output voltage VOUT in the frequency domain over five decades of frequencies (between 1 Hz and 100,000 Hz). Each decade isdivided into 20 parts (using a logarithmic scale), so that there are a total of 101 fitness cases (sampled frequencies).

The fitness of a circuit is defined in terms of two factors. The first factor measures the circuit's behavior in the frequency domain while the second factor measures the circuit's similarity to the to-be-avoided ladder filter.

The first factor is the sum, over the 101 fitness cases, of the absolute weighted deviation between the actual value of the voltage that is produced by the given circuit at the probe point VOUT and the target value for voltage (0 or 1 volts) for that frequency. Specifically, this factor 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 factor of the fitness measure pertaining to the circuit's frequency response does not penalize ideal voltage values, slightly penalizes acceptable voltage deviations, and heavily penalizes unacceptable voltage deviations. Specifically, if the output voltage equals the ideal value of 1.0 volt for each of the 61 points in the intended passband between 1 Hz and 1,000 Hz, 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 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 acceptable deviation of 1 millivolt from the ideal voltage of 0 volts. The deviations are deemed to be zero for each of the five "don't care" points between 1,000 and 2,000 Hz. The number of "hits" is defined as the number of fitness cases for which the voltage is acceptable, ideal, or that lie in the "don't care" band.

The factor of the fitness measure pertaining to similarity to the to-be-avoided ladder filter is measured in terms of the largest number of nodes and edges (circuit components) of a subgraph of the given circuit that is isomorphic to a subgraph of a template. The template is a Campbell ladder that is far larger than is needed to solve the problem at hand. As shown in figure 3, the template contains 16 shunt capacitors, 17 series inductors, the circuit's voltage source, and a source resistor.

The score is determined by a graph isomorphism algorithm (Ullman 1976, Lingas 1981) with the cost function here being based on the number of shared nodes and edges (instead of just the number of nodes). Since the graph isomorphism algorithm works with graph adjacency matrices and since circuits often contain parallel compositions of two different components, each pair of parallel components that is encountered is treated as if it were a type of component different from either a single capacitor or single inductor.

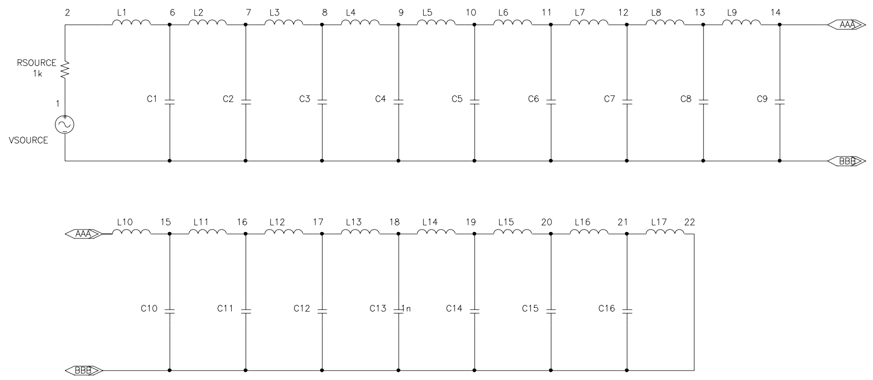

Figure 3 Campbell template.

For reference, the factor of the fitness measure pertaining to the frequency response of the 100%-compliant seven-rung ladder filter of figure 1 is 0.00784 (i.e., near zero) and its isomorphism factor is 25 (very high). The details of the calculation of the isomorphism score are found in Koza, Bennett, Andre, and Keane 1999.

A lower value of each of the above two factors of the fitness measure is better. For circuits not scoring the maximum number (101) of hits, the fitness of a circuit is the product of the two factors. For circuits scoring 101 hits (100%-compliant individuals), fitness is the number of shared nodes and edges divided by 10,000. This arrangement has the feature of almost always assigning a better (lower) fitness to any individual scoring 101 hits than to any individual scoring fewer than 101 hits.

A smaller the overall value of fitness is better. A fitness of zero is unattainable because every circuit has at least one node or edge in common with the template (even if only a single component).

Many of the random initial circuits and many that are created by crossover and mutation in subsequent generations are so pathological that the SPICE simulator cannot simulate them. These circuits receive a high penalty value of fitness (10^8) and become the worst-of-generation programs for each generation. In the run described below, 91% of the circuits in generation 0 cannot be simulated (compared to about 5% in later generations).

4.6 Control Parameters

The population size, *M*, is 1,950,000. The maximum size of each branch of each circuit-constructing program tree is 300 points (functions and terminals). Other control parameters are those used previously for the lowpass filter problem (Koza, Bennett, Andre, and Keane 1999, appendix D).

4.7 Termination Criterion and Results Designation

Since the goal is to generate a variety of 100%-compliant circuits for examination as to their novelty, the run was not automatically terminated upon evolution of the first

100%-compliant individual. Instead, numerous 100%-compliant circuits were harvested; and the run was manually monitored and manually terminated.

4.8 Implementation on Parallel Computer

This problem was run on a home-built Beowulf-style (Sterling, Salmon, Becker, and Savarese 1999) parallel cluster computer system consisting of 65 processing nodes (each containing a 533-MHz DEC Alpha microprocessor and 64 megabytes of RAM) arranged in a two-dimensional 5×13 toroidal mesh. The system has a DEC Alpha type computer as host. The processing nodes are connected with a 100 megabit-per-second Ethernet. The processing nodes and the host use the Linux operating system. The distributed genetic algorithm was used with a population size of $Q = 30{,}000$ at each of the $D = 65$ demes (semi-isolated subpopulations) for a total of population, M, of 1,950,000. Generations are asynchronous on the nodes. On each generation, four boatloads of emigrants, each consisting of $B = 2\%$ (the migration rate) of the node's subpopulation (selected probabilistically on the basis of fitness using the same selection procedure as used for the genetic operations) were dispatched to each of the four adjacent processing nodes (Koza, Bennett, Andre, and Keane 1999).

5 Results

A run starts with the random creation of an initial population (generation 0) of circuit-constructing program trees composed of the problem’s functions and terminals. The best-of-generation circuit from generation 0 (figure 4a) scores 52 hits (out of 101). The fitness of this best-of-generation circuit from generation 0 is 296.5 because the factor pertaining to this circuit’s frequency response is 59.30 and because this circuit’s isomorphism factor is 5. The isomorphism factor is 5 because the largest number of nodes and edges of a subgraph of this circuit that is isomorphic to a subgraph of the 17-inductor, 16-capacitor template (figure 3) consists of five nodes and edges.

Figure 4a shows the behavior in the frequency domain of the best circuit of generation 0. As can be seen, the behavior of this circuit bears little resemblance to the desired lowpass filter. 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 linearly between 0 to 1 Volts. The circuit delivers a full volt only for frequencies up to about 50 Hz. The circuit suppresses the incoming signal only for a handful of frequencies near 100,000 Hz. There is a very leisurely transition region in between. Circuits with improved fitness are evolved as the run proceeds from generation to generation.

The best-of-generation circuit from generation 16 (figure 5) has three inductors and four capacitors and scores 95 hits. Capacitors C1, C2, C3, and C4 are shunt capacitors and constitute the rungs of a ladder, while inductors L1, L4, and L3 (horizontally across the top of the figure) are the ladder’s series inductors.

This circuit constitutes the rediscovery by genetic programming of the well-known ladder topology of the Campbell filter. When this circuit is compared with the template, it is assigned a high (undesirable) isomorphism factor. The overall fitness of this circuit is 32.32 because the factor pertaining to this circuit’s frequency response is 2.694 and its isomorphism factor is 12.

Figure 4b shows the behavior in the frequency domain of the best circuit of generation 16. As can be seen, the behavior of this circuit bears some resemblance to the desired lowpass filter. Its transition region is more sharply defined than that of the best circuit of generation 0 (figure 4a).

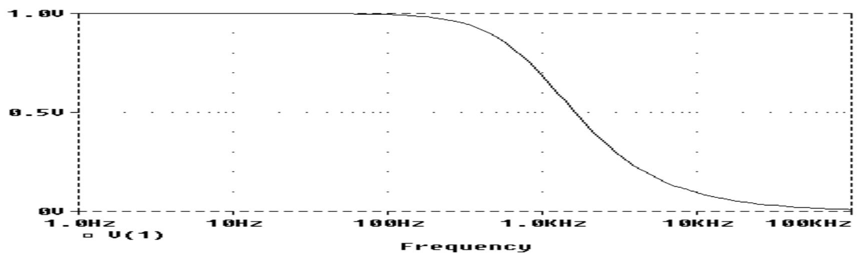

Figure 4a Frequency domain behavior of best circuit from generation 0.

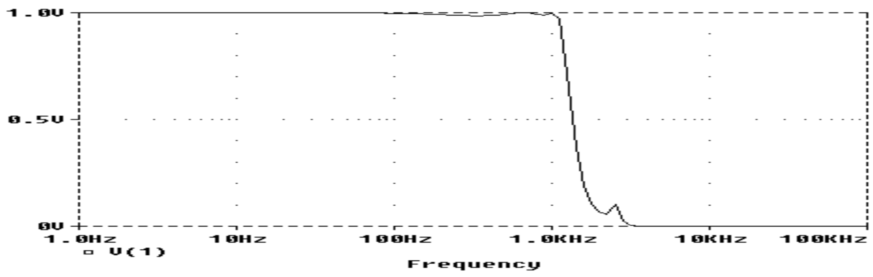

Figure 4b Frequency domain behavior of best circuit from generation 16.

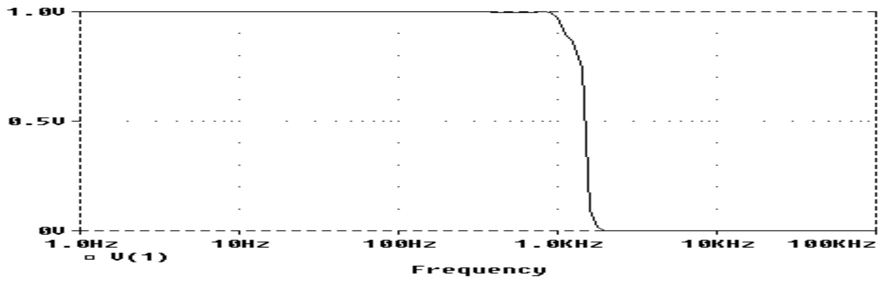

Figure 4c Frequency domain behavior of 100%-compliant circuit with elliptic topology.

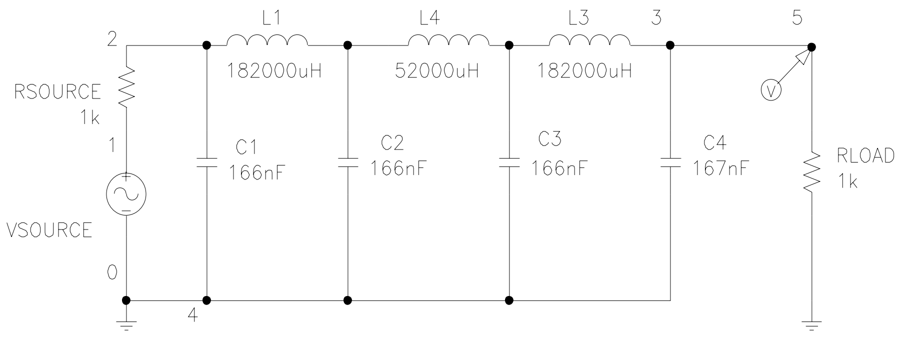

Figure 5 Best circuit of generation 16.

As the run progresses, some circuits score well primarily because of the factor pertaining to the circuit's frequency response while others score well primarily because of the isomorphism factor. For example, the fitness of one pace-setting circuit from generation 18 is 30.585. The factor of the fitness measure pertaining to this circuit's frequency response is 6.117 (not very good) but its isomorphism factor is 5 (reflecting great dissimilarity between this circuit and the 17-inductor, 16-capacitor template of figure 3). On the other hand, the fitness of another pace-setting circuit from generation 18 (in fact, the best-of-generation individual) is 11.556. The factor of the fitness measure pertaining to frequency response is 0.7704 (very good behavior in the frequency domain) but its isomorphism factor is 15 (reflecting great similarity between this circuit and the template).

As the run proceeds from generation to generation, circuits begin to appear that score well because of both the isomorphism factor and the factor pertaining to the circuit's frequency response. Eight different 100%-compliant circuits (i.e., circuits scoring 101 hits) were harvested from this run. Figure 4c shows the behavior in the frequency domain of the first 100%-compliant circuit evolved in this run; the behavior of the other seven circuits is similar. As can be seen in the figure, the circuit delivers nearly a full volt for frequencies up to 1,000 Hz; there is a very sharp drop-off between 1,000 Hz and 2,000 Hz; and the circuit effectively suppresses the output above 2,000 Hz. None of the 100%-compliant circuits harvested from this run have the ladder topology patented by Campbell in 1917. In other words, genetic programming evolved multiple novel solutions in this run to the given problem and each evolved solution avoided the prior art. Table 1 shows the factor pertaining to the circuit's frequency response, the isomorphism factor, and the overall fitness.

Table 1 Fitness of eight 100%-compliant circuits.

	Generation	Frequency response factor	Isomorphism factor	Overall fitness
1	13	0.051039	7	0.357273
2	14	0.117093	7	0.819651
3	14	0.103064	7	0.721448
4	15	0.161101	7	1.127707
5	15	0.044382	13	0.044382
6	15	0.133877	7	0.937139
7	16	0.059993	5	0.299965
8	13	0.062345	11	0.685795

Seven of the eight 100%-compliant circuits have highly irregular and asymmetric topologies. Figure 6 shows the chronologically first individual scoring 101 hits that appeared in the run. This circuit has an overall fitness of 0.685795. The factor of the fitness measure pertaining to the circuit's frequency response is 0.062345 while the isomorphism factor is 11. The result-producing branches of its circuit-constructing program tree contain 181 and 115 points, respectively. The circuit consists of four parallel compositions of an inductor and a capacitor (appearing horizontally across the top of the figure) and three shunt capacitors (appearing vertically in the figure).

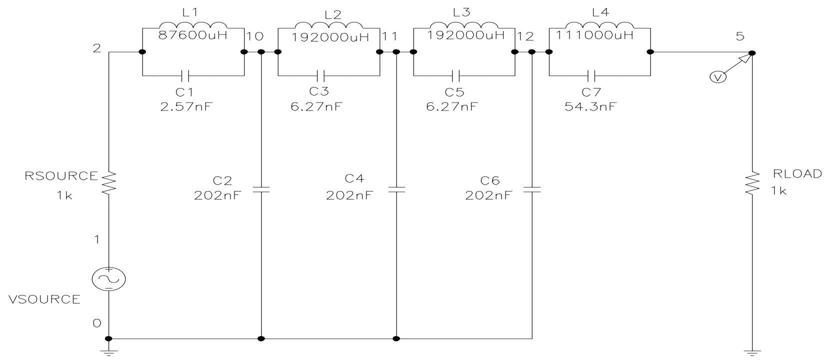

Figure 6 Evolved 100%-compliant circuit with elliptic topology.

Once genetic programming has successfully created one or more novel solutions to the given problem, a design engineer may examine them. Some may have unexpected virtues. The topology of the evolved filter of figure 6 is one form of the elliptic filter that Cauer invented and patented (Cauer 1934, 1935, 1936). The elliptic topology was invented and patented by Cauer in response to a long-standing need in the telephone industry for filters that were less expensive to manufacture. At the time of its invention by Cauer, the elliptic filter was a significant advance (both theoretically and commercially) over the prior art of their time, namely the then-known Campbell filter (and the closely related Butterworth and Chebychev filters). Specifically, for one commercially important set of specifications for the telephone industry, a fifth-order elliptic filter matches the behavior of a 17th-order Butterworth filter or an eighth-order Chebychev filter. The fifth-order elliptic filter has one less component than the eighth-order Chebychev filter. This reduction was very important in the days when filters in telephones were manufactured by individually soldering in expensive discrete components. As Van Valkenburg (1982, page 379) relates in connection with the history of the elliptic filter:

> Cauer first used his new theory in solving a filter problem for the German telephone industry. His new design achieved specifications with one less inductor than had ever been done before. The world first learned of the Cauer method not through scholarly publication but through a patent disclosure, which eventually reached the Bell Laboratories. Legend has it that the entire Mathematics Department of Bell Laboratories spent the next two weeks at the New York Public library studying elliptic functions. Cauer had studied mathematics under Hilbert at Goettingen, and so elliptic functions and their applications were familiar to him.

The elliptic topology invented and patented by Cauer was reinvented by genetic programming in this run as a consequence of the fact that the fitness measure rewarded candidate solutions that were dissimilar to the previously known Campbell filter topology. Cauer received a patent for the elliptic filter because his design satisfied the legal criteria for obtaining a U. S. patent, including the fact that it was "new” and “useful" and

> ... the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would [not] have been

obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. (35 *United States Code* 103a).

6 Conclusion

We have established the principle that genetic programming can automatically create designs that satisfactorily solve a problem while simultaneously avoiding prior art. The reinvention by genetic programming of the patented Campbell and Cauer topologies for filters is an instance where genetic programming has produced a result that is competitive with a result created by a creative and inventive human. These evolved results satisfy Arthur Samuel's criterion (1983) for artificial intelligence and machine learning, namely

The aim [is] ... to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence.

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.

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.

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

Cauer, Wilhelm. 1934. *Artificial Network*. U. S. Patent 1,958,742. Filed June 8, 1928 in Germany. Filed December 1, 1930 in United States. Issued May 15, 1934.

Cauer, Wilhelm. 1935. *Electric Wave Filter*. U. S. Patent 1,989,545. Filed June 8, 1928. Filed December 6, 1930 in United States. Issued January 29, 1935.

Cauer, Wilhelm. 1936. *Unsymmetrical Electric Wave Filter*. Filed November 10, 1932 in Germany. Filed November 23, 1933 in United States. Issued July 21, 1936.

Gruau, Frederic. 1992. *Cellular Encoding of Genetic Neural Networks*. Technical report 92-21. Laboratoire de l'Informatique du Parallélisme. Ecole Normale Supérieure de Lyon. May 1992.

Gruau, Frederic. 1994a. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD Thesis. Ecole Normale Supérieure de Lyon.

Gruau, Frederic. 1994b. Genetic micro programming of neural networks. In Kinnear, Kenneth E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press. Pages 495–518.

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.

Kitano, Hiroaki. 1990. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*. 4(1990) 461–476.

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). *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*. Cambridge, MA: The MIT Press. Pages 695–717.

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 Dunlap, Frank. 1997. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*. 1(2). Pages 109 – 128.

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* 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*. Cambridge, MA: The MIT Press.

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

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

Lingas, Andrzej. 1981. Certain algorithms for subgraph isomorphism problems. In Astesiano, E. and Bohm, C. (editors). *Proceedings of the. Sixth Colloquium on Trees in Algebra and Programming***.** Lecture Notes on Computer Science. Springer Verlag. Volume 112. Pages 290 – 307.

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.

Samuel, Arthur L. 1983. AI: Where it has been and where it is going. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann. Pages 1152 – 1157.

Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter (editors). 1999. *Advances in Genetic Programming 3*. Cambridge, MA: The MIT Press.

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

Ullman, J. R. 1976. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*. 23(1) 31 – 42. January 1976.

Williams, Arthur B. and Taylor, Fred J. 1995. *Electronic Filter Design Handbook*. Third Edition. New York, NY: McGraw-Hill.

Version 2 – Submitted February 19, 1999 to Euro-GP Workshop to be held in Goteborg, Sweden on May 26 – 27, 1999.

Genetic Programming as a Darwinian Invention Machine

John R. Koza
Section on Medical Informatics
Department of Medicine
Stanford, California 94305
koza@stanford.edu
http://www.smi.stanford.edu/people/koza/

Forrest H Bennett III
Chief Scientist
Genetic Programming Inc.
Box 1669
Los Altos, California 94023
forrest@evolute.com

Oscar Stiffelman
Computer Science Department
Stanford University
Stanford, California 94305
ozzie@cs.stanford.edu