Routine Human-Competitive Machine Intelligence by Means of Genetic Programming

John R. Koza*[a], Matthew J. Streeter[b], Martin A. Keane[c]
[a]Stanford University, Stanford, CA, USA 94305
[b]Genetic Programming Inc., Mountain View, CA, USA 94041
[c]Econometrics Inc., Chicago, IL, USA 60611

ABSTRACT

Genetic programming is a systematic method for getting computers to automatically solve a problem. Genetic programming starts from a high-level statement of what needs to be done and automatically creates a computer program to solve the problem. The paper demonstrates that genetic programming (1) now routinely delivers high-return human-competitive machine intelligence; (2) is an automated invention machine; (3) can automatically create a general solution to a problem in the form of a parameterized topology; and (4) has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time. Recent results involving the automatic synthesis of the topology and sizing of analog electrical circuits and controllers demonstrate these points.

Keywords: Genetic programming, automated inventions, parameterized topology, analog circuit synthesis, controller synthesis

1 INTRODUCTION

One of the central challenges of computer science is to get a computer to solve a problem without explicitly programming it to do so. Paraphrasing Arthur Samuel—founder of the field of machine learning—this challenge (1959) is

> How can computers be made to do what needs to be done, without being told exactly how to do it?

Samuel described the criterion for success for the above challenge in his 1983 talk entitled "AI: Where It Has Been and Where It Is Going":

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

Genetic programming starts from a high-level statement of what needs to be done and automatically creates a computer program to solve the problem. Genetic programming uses the Darwinian principle of natural selection along with analogs of recombination (crossover), mutation, gene duplication, gene deletion, and mechanisms of developmental biology to breed an ever-improving population of programs.

This paper makes four points:

(1) Genetic programming now routinely delivers high-return human-competitive machine intelligence (section 3).

(2) Genetic programming is an automated invention machine (section 4).

(3) Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology (section 5).

(4) Genetic programming has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time (section 6).

*koza@stanford.edu; phone 650-941-0336; http://www.smi.stanford.edu/people/koza.

2 BACKGROUND ON GENETIC PROGRAMMING

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

(1) Generate an initial set (called the *population*) of compositions (typically random) of functions and terminals appropriate to the problem.

(2) Iteratively perform the following substeps (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 problem's fitness measure.

(B) Create a new population (the next generation) of programs by applying the following operations 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:* Create one new offspring program for the new population by applying a selected architecture-altering operation to the selected program.

(3) Designate the individual program that is identified by the result designation method (e.g., the individual with the best fitness) as the run's result. This result may be a solution (or approximate solution) to the problem.

Genetic programming has been successfully applied to a wide variety of problems from numerous different fields (Koza 1992; Koza and Rice 1992; Koza 1994a, 1994b; Koza, Bennett, Andre, and Keane 1999; Koza, Bennett, Andre, Keane, and Brave 1999; Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003; Koza, Keane, Streeter, Mydlowec, Yu, Lanza, and Fletcher 2003).

3 GENETIC PROGRAMMING NOW ROUTINELY DELIVERS HIGH-RETURN HUMAN-COMPETITIVE MACHINE INTELLIGENCE

3.1 Definition of "Human-Competitive"

In attempting to evaluate an automated problem-solving method, the question arises as to whether there is any real substance to the demonstrative problems that are published in connection with the method. Demonstrative problems in the fields of artificial intelligence and machine learning are often contrived toy problems that circulate exclusively inside academic groups that study a particular methodology and that have little relevance to any issues pursued by other scientists or engineers.

Table 1 Eight criteria for human-competitiveness

	Criterion
A	The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
B	The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.
C	The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
D	The result is publishable in its own right as a new scientific result—independent of the fact that the result was mechanically created.
E	The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
F	The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
G	The result solves a problem of indisputable difficulty in its field.
H	The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

Table 2 Thirty-six human-competitive results produced by genetic programming

	Claimed instance	Basis
1	Creation of a better-than-classical quantum algorithm for the Deutsch-Jozsa "early promise" problem	B, F
2	Creation of a better-than-classical quantum algorithm for Grover's database search problem	B, F
3	Creation of a quantum algorithm for depth-two AND/OR query problem that is better than previously published results	D
4	Creation of a quantum algorithm for the depth-one OR query problem that is better than any previously published result	D
5	Creation of a protocol for communicating information through a quantum gate previously thought not to permit it	D
6	Creation of a novel variant of quantum dense coding	D
7	Creation of a soccer-playing program that won its first two games in the Robo Cup 1997 competition	H
8	Creation of a soccer-player ranked in middle of field of 34 human-written programs in Robo Cup 1998 competition	H
9	Creation of four different algorithms for the transmembrane segment identification problem for proteins	B, E
10	Creation of a sorting network for seven items using only 16 steps	A, D
11	Rediscovery of the Campbell ladder topology for lowpass and highpass filters	A, F
12	Rediscovery of the Zobel "*M*-derived half section" and "constant *K*" filter sections	A, F
13	Rediscovery of the Cauer (elliptic) topology for filters	A, F
14	Automatic decomposition of the problem of synthesizing a crossover filter	A, F
15	Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of amplifier and other circuits	A, F
16	Synthesis of 60 and 96 decibel amplifiers	A, F
17	Synthesis of analog computational circuits for squaring, cubing, square root, cube root, and Gaussian functions	A, D, G
18	Synthesis of a real-time analog circuit for time-optimal control of a robot	G
19	Synthesis of an electronic thermometer	A, G
20	Synthesis of a voltage reference circuit	A, G
21	Creation of a cellular automata rule for the majority classification problem that is better than the Gacs-Kurdyumov-Levin (GKL) rule and all other known rules written by humans	D, E
22	Creation of motifs that detect the D–E–A–D box family of proteins and the manganese superoxide dismutase family	C
23	Synthesis of topology for a PID-D2 (proportional, integrative, derivative, and second derivative) controller	A, F
24	Synthesis of an analog circuit equivalent to Philbrick circuit	A, F
25	Synthesis of a `NAND` circuit	A, F
26	Simultaneous synthesis of topology, sizing, placement, and routing of analog electrical circuits	A. F, G
27	Synthesis of topology for a PID (proportional, integrative, and derivative) controller	A, F
28	Rediscovery of negative feedback	A, E, F, G
29	Synthesis of a low-voltage balun circuit	A
30	Synthesis of a mixed analog-digital variable capacitor circuit	A
31	Synthesis of a high-current load circuit	A
32	Synthesis of a voltage-current conversion circuit	A
33	Synthesis of a Cubic function generator	A
34	Synthesis of a tunable integrated active filter	A
35	Creation of PID tuning rules that outperform the Ziegler-Nichols and Åström-Hägglund tuning rules	A, B, D, E, F, G
36	Creation of non-PID controllers that outperform the Ziegler-Nichols or Åström-Hägglund tuning rules	A, B, D, E, F, G

To make the idea of human-competitiveness concrete, we say that a result is "human-competitive" if it satisfies one or more of the eight criteria in table 1. These eight criteria have the desirable attribute of being at arms-length from the fields of artificial intelligence, machine learning, and genetic programming. That is, a result cannot acquire the rating of "human-competitive" merely because it is considered interesting by researchers *inside* the specialized fields that are

attempting to create machine intelligence. Instead, a result produced by an automated method must earn the rating of "human-competitive" *independent* of the fact that it was generated by an automated method.

Based on this definition, there are now 36 instances where genetic programming has produced a human-competitive result (table 2).

3.2 Definition of "High-Return"

What is delivered by the actual automated operation of an artificial method in comparison to the amount of knowledge, information, analysis, and intelligence that is pre-supplied by the human employing the method?

We define the *AI ratio* (the "artificial-to-intelligence" ratio) of a problem-solving method as the ratio of that which is delivered by the automated operation of the *artificial* method to the amount of *intelligence* that is supplied by the human applying the method to a particular problem.

The AI ratio is especially pertinent to methods for getting computers to automatically solve problems because it measures the value added by the artificial problem-solving method. Manifestly, the aim of the fields of artificial intelligence and machine learning is to generate human-competitive results with a high AI ratio.

Ascertaining the return of a problem-solving method requires measuring the amount of "A" that is delivered by the method in relation to the amount of "I" that is supplied by the human user.

Each of the 36 results in table 2 is a human-competitive result. Therefore, it is reasonable to say that genetic programming delivered a high amount of "A" for each of them.

The question thus arises as to how much "I" was supplied by the human user in order to produce these 36 results. Answering this question requires the discipline of carefully identifying the amount of analysis, intelligence, information, and knowledge that was supplied by the intelligent human user prior to launching a run of genetic programming.

To do this, we make a clear distinction between the problem-specific preparatory steps and the problem-independent executional steps of a run of genetic programming.

The *preparatory steps* are the problem-specific and domain-specific steps that are performed by the human user prior to launching a run of the problem-solving method. The preparatory steps establish the "I" component of the AI ratio (i.e., the denominator).

The *executional steps* are the problem-independent and domain-independent steps that are automatically executed during a run of the problem-solving method. The results produced by the executional steps provide the "A" component of the AI ratio (i.e., the numerator).

The five major preparatory steps for genetic programming require the human user to specify

(1) the set of terminals (e.g., the independent variables of the problem, zero-argument functions, and random constants) available to each branch of the to-be-evolved computer program,

(2) the set of primitive functions available to each branch of the to-be-evolved computer program,

(3) the fitness measure (for explicitly or implicitly measuring the fitness of individuals in the population),

(4) certain parameters for controlling the run, and

(5) a termination criterion and method for designating the result of the run.

For the human-competitive results in table 2, only a *de minimus* amount of "I" is contained in the human-supplied primitive ingredients available to the to-be-evolved computer program (the first and second preparatory steps), the human-

supplied fitness measure (the third preparatory step containing the high-level statement of what needs to be done), and the human-supplied control parameters and termination procedures (the fourth and fifth preparatory steps). In any event, the amount of "I" required by genetic programming is certainly not greater than that required by any other method of artificial intelligence and machine learning of which we are aware. Indeed, we know of no other problem-solving method (automated or human) that does not start with primitive elements of some kind, that does not employ some method for specifying what needs to be done to guide or evaluate the method's operation, that does not employ some kind of administrative parameters, and that does not employ some kind of termination criterion.

In view of the high amount of "A" in the numerator and the small amount of "I" in the denominator, we can see that the AI ratio is high for the results in table 2 produced by genetic programming..

3.3 Definition of "Routine"

Generality is a precondition to what we mean when we say that an automated problem-solving method is "routine." Once the generality of a method is established, "routineness" means that relatively little human effort is required to get the method to successfully handle new problems within a particular domain and to successfully handle new problems from a different domain. The ease of making the transition to new problems lies at the heart of what we mean by "routine."

For example, virtually all controllers are built from the same primitive ingredients (e.g., integrators, differentiators, gains, adders, subtractors, and signals representing the plant output and the reference signal). Once these primitive ingredients are identified, new problems of controller synthesis can be handled merely by changing the statement of what needs to be done. Thus, after solving one problem of automatically synthesizing both the topology and tuning of a controller (say, item 27 in table 2), the transition to each new problem of controller synthesis (say, item 23) merely involves providing genetic programming with a different specification of what needs to be done—that is, a different fitness measure.

In making the transition from problems of automatic synthesis of controllers to problems of automatic synthesis of circuits, the primitive ingredients change from integrators, differentiators, gains, and the like to transistors, resistors, capacitors, and the like. The fitness measure changes from one that minimizes a controller's integral of time-weighted absolute error, minimizes overshoot, and maximizes disturbance rejection to one that is based on the circuit's amplification, suppression or passage of a signal, elimination of distortion, and the like. That is, the transition from problem to problem is routine when using genetic programming.

4 GENETIC PROGRAMMING IS AN AUTOMATED INVENTION MACHINE

There are now 23 instances where genetic programming has duplicated the functionality of a previously patented invention, infringed a previously issued patent, or created a patentable new invention. Specifically, there are 15 instances where genetic programming has created an entity that either infringes or duplicates the functionality of a previously patented 20^{th}-century invention, six instances where genetic programming has done the same with respect to a previously patented 21^{st}-century invention, and two instances where genetic programming has created a patentable new invention.

To make the foregoing points concrete, this section presents six instances where genetic programming automatically created both the topology (graphical structure) and sizing (numerical component values) for patented analog electrical

circuits composed of transistors, capacitors, resistors, and other components. In each instance, genetic programming started from a high-level statement of a circuit's desired behavior and characteristics (e.g., its desired output given its input). In producing results, genetic programming used only *de minimus* knowledge about analog circuits. Specifically, genetic programming employed a circuit simulator (e.g., SPICE) for the *analysis* of candidate circuits, but did not use any deep knowledge or expertise about the *synthesis* of circuits. The six inventions are show in table 3.

When genetic programming is used to automatically create computer programs, the programs are ordinarily represented as trees (i.e., rooted, point-labeled trees with ordered branches). In contrast, electrical circuits are usually represented as labeled graphical structures in which each component is included in a cycle. Thus, there is a representational obstacle that must be overcome before genetic programming can be applied to the problem of automatically synthesizing circuits. This obstacle can be overcome by establishing a mapping between program trees and labeled cyclic graphs. The mapping from trees into circuits is accomplished by means of a developmental process. This process begins with a simple embryo. The embryo herein consists of a single isolated modifiable wire that is not initially connected to external inputs or outputs. An analog electrical circuit is developed by progressively applying the functions in a circuit-constructing program tree to the embryo's initial modifiable wire (and to succeeding modifiable wires and modifiable components).

Table 3 Six post-2000 patented analog circuits

Invention	Inventor(s) and date	Institution
Low-voltage balun circuit	Sang Gug Lee (2001)	Information and Communications University
Mixed analog-digital variable capacitance	Turgut Sefket Aytur (2002)	Lucent Technologies Inc.
Voltage-current converter	Akira Ikeuchi and Naoshi Tokuda (2000)	Mitsumi Electric Co., Ltd.
High-current load circuit for testing a voltage source	Timothy Daun-Lindberg and Michael Miller (2001)	International Business Machines Corporation
Low-Voltage cubic function generator	Stefano Cipriani and Anthony A. Takeshian (2000)	Conexant Systems, Inc.
Tunable integrated active filter	Robert Irvine and Bernd Kolb (2001)	Infineon Technologies AG

The available functions in a circuit-constructing program tree include

(1) topology-modifying functions that alter the topology of a developing circuit (e.g., series division, parallel division, via between nodes, via to ground, via to a power supply, via to input, via to output),

(2) component-creating functions that insert components (i.e., resistors, capacitors, and transistors) into a developing circuit, and

(3) development-controlling functions that control the developmental process (e.g., cut, end).

The function and terminal sets for all six problems permit the construction of any circuit composed of transistors, resistors, and capacitors.

The main difference among the runs of genetic programming for the six problems (described below) is that we supplied a different fitness measure for each problem. Construction of a fitness measure requires translating the problem's high-level requirements into a precise computation. We read the patent document to find the performance that the invention was supposed to achieve. We then created a fitness measure reflecting the invention's performance and characteristics. The fitness measure specifies the time-domain output value(s) that is desired given various time-domain input value(s). For each problem, a test fixture consisting of certain fixed components (such as a source resistor, a load resistor) is connected to the desired input port(s) and the desired output port(s). Circuits are simulated using SPICE.

We supplied models for transistors appropriate to the problem. We used the commercially common 2N3904 (*npn*) and 2N3906 (*pnp*) transistor models unless the patent document called for a different model. We used 5-Volt power supplies unless the patent specified otherwise.

The control parameters and termination criterion were the same for all six problems, except that we used different population sizes to approximately equalize each run's elapsed time per generation.

We now describe the six fitness measures. For additional details, see Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003.

4.1 Fitness Measures for the Six Problems

4.1.1 Low-Voltage Balun Circuit

The purpose of a balun (balance/unbalance) circuit is to produce two outputs from a single input, each output having half the amplitude of the input, one output being in phase with the input while the other is 180 degrees out of phase with the input, with both outputs having the same DC offset. The patented balun circuit uses a power supply of only 1 Volt. The fitness measure consisted of (1) a frequency sweep analysis designed to ensure the correct magnitude and phase at the two outputs of the circuit and (2) a Fourier analysis designed to penalize harmonic distortion.

4.1.2 Mixed Analog-Digital Register-Controlled Variable Capacitor

This mixed analog-digital circuit has a capacitance that is controlled by the value stored in a digital register. The fitness measure employed 16 time-domain fitness cases. The 16 fitness cases ranged over all eight possible values of a 3-bit digital register for two different analog input signals.

4.1.3 Voltage-Current Conversion Circuit

The purpose of the voltage-current conversion circuit is to take two voltages as input and to produce a stable current whose magnitude is proportional to the difference of the voltages. We employed four time-domain input signals (fitness cases) in the fitness measure. We included a time-varying voltage source beneath the output probe point to ensure that the output current produced by the circuit was stable with respect to any subsequent circuitry to which the output of the circuit might be attached.

4.1.4 High-Current Load Circuit

The patent covers a circuit designed to sink a time-varying amount of current in response to a control signal. The patented circuit employs a number of FET transistors arranged in parallel, each of which sinks a small amount of the desired current. The fitness measure consisted of two time-domain simulations, each representing a different control signal.

4.1.5 Low-Voltage Cubic Signal Generator

The patent covers an analog computational circuit that produces the cube of an input signal as its output. The circuit is "compact" in that it contains a voltage drop across no more than two transistors.

The fitness measure for this problem consisted of four time-domain fitness cases using various input signals and time scales. The compactness constraint was enforced by providing only a 2-Volt power supply.

4.1.6 Tunable Integrated Active Filter

The patent covers a tunable integrated active filter that performs the function of a lowpass filter whose passband boundary is dynamically specified by a control signal. The circuit has two inputs: a to-be-filtered incoming signal and a control signal.

The fitness measure for this problem consisted of a performance penalty and a parsimony penalty. The passband boundary, f, ranges over nine values between 441 and 4,414 Hz. The performance penalty is a weighted sum, over 61

frequencies for each of the nine values of *f*, of the absolute weighted deviation between the output of the individual candidate circuit at its probe point and the target output. The parsimony penalty is equal to the number of components in the circuit. For additional details, see Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003.

4.2 Results for the Six Problems

4.2.1 Low-Voltage Balun Circuit

Genetic programming automatically created the circuit shown in figure 1. This best-of-run evolved circuit was produced in generation 97 and has a fitness of 0.429. The patented circuit has a fitness of 1.72. That is, the evolved circuit is roughly a fourfold improvement (less being better) over the patented circuit in terms of our fitness measure. In addition, the evolved circuit is superior to the patented circuit both in terms of its frequency response and its harmonic distortion.

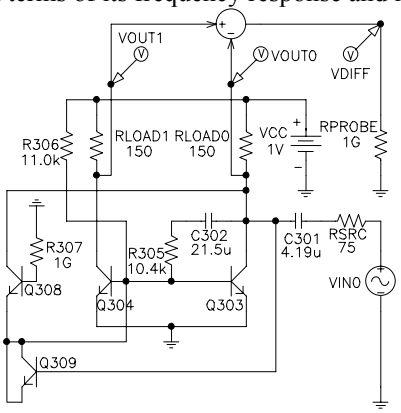

Figure 1 Best-of-run balun circuit

In the patent documents, Lee (2001) shows a previously known conventional (prior art) balun circuit. This previously known circuit is shown as figure 2 herein.

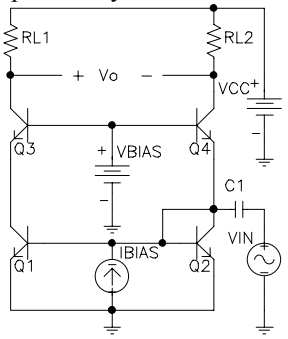

Figure 2 A prior art balun circuit shown in U.S. patent 6,265,908

Lee's patented low-voltage balun circuit is shown in figure 3 of this paper. Lee (2001) states that the essential difference between the prior art and his invention is a coupling capacitor C_2 located between the base and the collector of the transistor Q_2. Lee explains the essence of his invention as follows:

> "The structure of the inventive balun circuit shown in [Figure 3] is identical to that of [Figure 2] except that a capacitor C_2 are further provided thereto. The capacitor C_2 is a coupling capacitor disposed between the base and the collector of the transistor Q_2 and serves to block DC components which may be fed to the base of the transistor Q_2 from the collector of the transistor Q_2."

As can be seen, the best-of-run genetically evolved circuit (figure 1) possesses the very feature that Lee identifies as the essence of his invention, namely the coupling capacitor called "C_{302}" in figure 1 and called "C_2" in figure 3.

The genetically evolved circuit also reads on three additional elements of claim 1 of Lee's 2001 patent. However, as it happens, the genetically evolved circuit does not infringe Lee's patent because it does not read on other elements enumerated in claim 1.

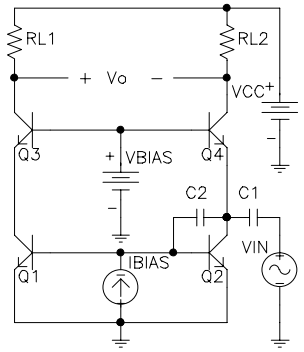

Figure 3 Lee's low voltage balun circuit shown in patent 6,265,908

4.2.2 Mixed Analog-Digital Register-Controlled Variable Capacitor

Over our 16 fitness cases, the patented circuit has an average error of 0.803 millivolts. In generation 95, a circuit emerged with average error of 0.808 millivolts, or approximately 100.6% of the average error of the patented circuit. During the course of this run, we harvested the smallest individuals produced on each processing node with a certain maximum level of error. Examination of these harvested individuals revealed a circuit from generation 98 (figure 4) that approximately matches the topology of the patented circuit (without infringing). The genetically evolved circuit reads on all but one of the elements of claim 1 of the patented circuit (and hence does not infringe the patent).

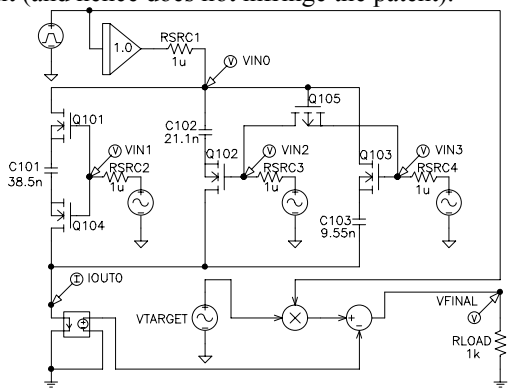

Figure 4 Evolved compliant register-controlled capacitor circuit

4.2.3 Voltage-Current Conversion Circuit

A circuit emerged on generation 109 of our run of this problem with a fitness of 0.619. That is, the evolved circuit has roughly 62% of the average (weighted) error of the patented circuit. The evolved circuit was subsequently tested on unseen fitness cases that were not part of the fitness measure and outperformed the patented circuit on these new fitness cases. The best-of-run circuit solves the problem in a different manner than the patented circuit.

4.2.4 High-Current Load Circuit

On generation 114, a circuit emerged that duplicated Daun-Lindberg and Miller's parallel FET transistor structure. This circuit has a fitness (weighted error) of 1.82, or 182% of the weighted error for the patented circuit.

The genetically evolved circuit shares the following features found in claim 1 of U.S. patent 6,211,726:

> "A variable, high-current, low-voltage, load circuit for testing a voltage source, comprising: …
>
> "a plurality of high-current transistors having source-to-drain paths connected in parallel between a pair of terminals and a test load."

However, the remaining elements of claim 1 in U.S. patent 6,211,726 are very specific and the genetically evolved circuit does not read on these remaining elements. In fact, the remaining elements of the genetically evolved circuit bear hardly any resemblance to the patented circuit. In this instance, genetic programming produced a circuit that duplicates the functionality of the patented circuit using a different structure.

4.2.5 Low-Voltage Cubic Signal Generator

The best-of-run evolved circuit (figure 5) was produced in generation 182 and has an average error of 4.02 millivolts. The patented circuit had an average error of 6.76 millivolts. That is, the evolved circuit has approximately 59% of the error of the patented circuit over our four fitness cases.

The claims in U.S. patent 6,160,427 amount to a very specific description of the patented circuit. The genetically evolved circuit does not read on these claims and, in fact, bears hardly any resemblance to the patented circuit. In this instance, genetic programming produced a circuit that duplicates the functionality of the patented circuit and does so using a very different structure.

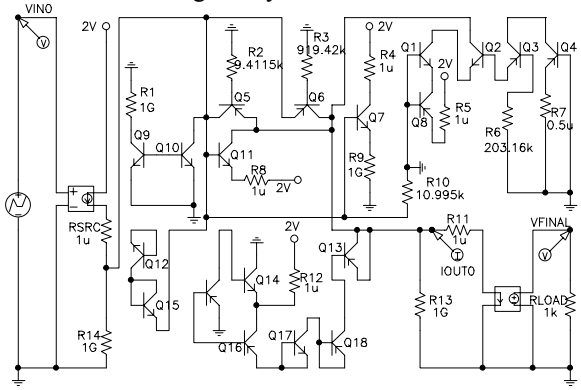

Figure 5 Best-of-run cubic signal generation circuit

4.2.6 Tunable Integrated Active Filter

Averaged over the nine values of frequency, the best-of-run circuit from generation 50 (figure 6) has 72.7 millivolts average absolute error for frequencies in the passband and 0.39 dB average absolute error for other frequencies.

The best-of-run genetically evolved circuit reads on every element of claim 1 of U.S. patent 6,225,859 and therefore infringes the patent.

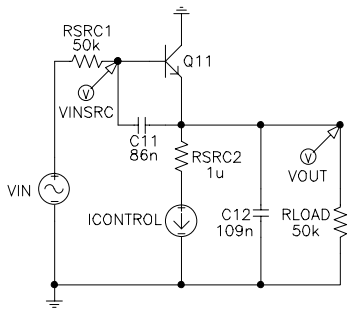

Figure 6 Best-of-run circuit for the tunable integrated active filter.

5 GENETIC PROGRAMMING CAN AUTOMATICALLY CREATE PARAMETERIZED TOPOLOGIES

Genetic programming can automatically create, in a single run, a general (parameterized) solution to a problem in the form of a graphical structure whose nodes or edges represent components and where the parameter values of the components are specified by mathematical expressions containing free variables. We call such a solution a *parameterized topology*.

In a parameterized topology, the genetically evolved graphical structure represents a complex structure (e.g., electrical circuit, controller, network of chemical reactions, antenna, genetic network). In the automated process, genetic programming determines the graph's size (its number of nodes) as well as the graph's connectivity (specifying which nodes are connected). Genetic programming also assigns, in the automated process, component types to the graph's nodes or edges. In the automated process, genetic programming also creates mathematical expressions that establish the parameter values of the components (e.g., the capacitance of a capacitor in a circuit). Some of these genetically created mathematical expressions contain free variables. The free variables confer generality on the genetically evolved solution by enabling a single genetically evolved graphical structure to represent a general (parameterized) solution to an entire category of problems. Genetic programming can do all the above in an automated way in a *single* run.

The capability of genetic programming to create parameterized topologies for design problems is illustrated by the automatic creation of a general-purpose non-PID controller (figure 7) whose blocks are parameterized by mathematical expressions containing the problem's four free variables, namely the plant's time constant, T_r, ultimate period, T_u, ultimate gain, K_u, and dead time, L. This genetically evolved controller (figure 7) outperforms PID controllers tuned using the widely used Ziegler-Nichols tuning rules (1942) and the recently developed Åström and Hägglund tuning rules (1995) on an industrially representative set of plants. The authors have applied for a patent on this new controller. For details, see Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003.

This controller's overall topology consists of three adders, three subtractors, four gain blocks parameterized by a constant, two gain blocks parameterized by non-constant mathematical expressions containing free variables, and two lead blocks parameterized by non-constant mathematical expressions containing free variables. Gain block 730 of figure 7 has a gain that is parameterized by the following non-constant mathematical expression (equation 31):

$$\left| \log \left| T_r - T_u + \log \left| \frac{\log\left(|L|^L\right)}{T_u + 1} \right| \right| \right| \qquad [31]$$

and gain block 760 of figure 7 has a gain that is parameterized by

$$\left|\log\left|T_r + 1\right|\right| \qquad [34].$$

Lead block 740 of figure 7 is parameterized by the following non-constant mathematical expression:

$$NLM\left(\log|L| - \left(\text{abs}(L)^L\right)^2 T_u^{\,3}\left(T_u + 1\right) T_r e^L - 2T_u e^L\right) \qquad [32].$$

In equation 32 (and 33 below), *NLM* is the nonlinear mapping

$$NLM(x) = \begin{cases} 10^0 & \text{if } x < -100 \\ 10^{-\frac{100}{19} - \frac{1}{19}x} & \text{if } -100 \le x < -5 \\ 10^x & \text{if } -5 \le x \le 5 \\ 10^{\frac{100}{19} - \frac{1}{19}x} & \text{if } 5 < x \le 100 \\ 10^0 & \text{if } x > 100. \end{cases}$$

Lead block 750 in figure 7 is parameterized by equation 33:

$$NLM\left(\log|L| - 2T_u e^L\left(2K_u\left(\log\left|K_u e^L\right| - \log|L|\right)T_u + K_u e^L\right)\right) \qquad [33].$$

If the genetically evolved program contains conditional developmental operators as well as free variables, a different graphical structure will, in general, be produced for different instantiations of the free variables. That is, the genetically evolved program operates as a genetic switch. Each program is provided with the problem's free variables as input. These values trigger the development of different graphical structures as the program is executed.

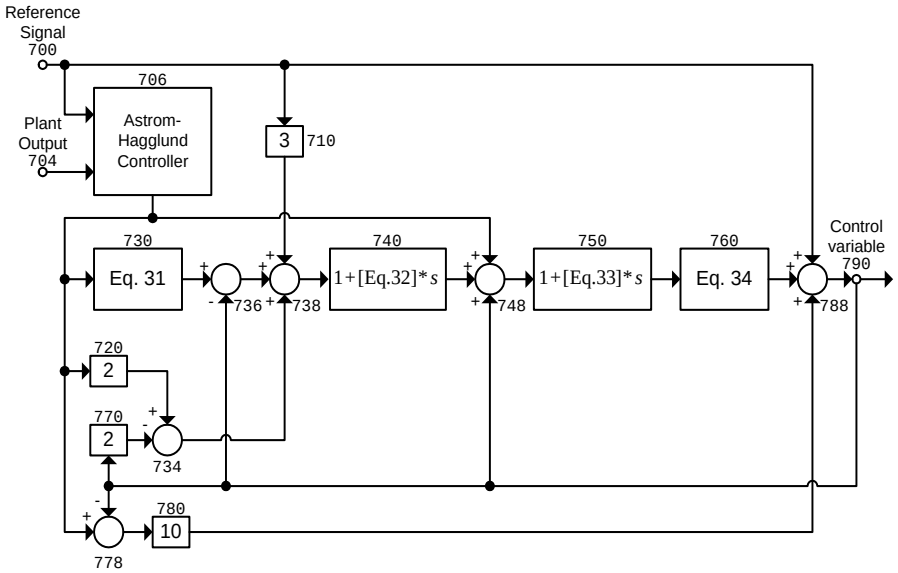

Figure 7 Parameterized topology for a general-purpose controller.

6 PROGRESSION OF QUALITATIVELY MORE SUBSTANTIAL RESULTS PRODUCED IN SYNCHRONY WITH INCREASING COMPUTER POWER

Table 4 lists the five computer systems used to produce our group's reported work on genetic programming in the 15-year period between 1987 and 2002. Column 7 shows the number of human-competitive results (as itemized in table 2) generated by each computer system.

The first entry in table 4 is a serial computer. The four subsequent entries are parallel computer systems. The presence of four increasingly powerful parallel computer systems in table 4 reflects the fact that genetic programming has successfully taken advantage of the increased computational power available by means of parallel processing (thereby avoiding a pitfall that often constrains other proposed approaches to machine intelligence).

Table 4 shows the following:

- There is an order-of-magnitude speed-up (column 4) between each successive computer system in the table. Note that, according to Moore's law, exponential increases in computer power correspond approximately to constant periods of time.
- There is a 13,900-to-1 speed-up (column 5) between the fastest and most recent machine (the 1,000-node parallel computer system) and the slowest and earliest computer system in table 4 (the serial LISP machine).
- The slower early machines generated few or no human-competitive results, whereas the faster more recent machines have generated numerous human-competitive results.

Four successive order-of-magnitude increases in computer power are explicitly shown in table 4. An additional order-of-magnitude increase was achieved by making extraordinarily long runs on the largest machine in table 4 (the 1,000-node Pentium® II parallel machine). The length of the run that produced the genetically evolved controller (figure 7) was 28.8 days—almost an order-of-magnitude increase (9.3 times) over the 3.4-day average for runs that our group has made in recent years. A patent application was filed for the controller produced by this four-week run. If this final 9.3-to-1 increase is counted as an additional speed-up, the overall speed-up is 130,660-to-1.

Table 4 Human-competitive results produced by genetic programming with five computer systems

System	Period	Petacycles (10^{15}) per day for system	Speed-up over previous row	Speed-up over first system in this table	Used for work in book	Human-competitive results
Serial Texas Instruments LISP machine	1987–1994	0.00216	1 (base)	1 (base)	*Genetic Programming I* and *Genetic Programming II*	0
64-node Transtech transputer parallel machine	1994–1997	0.02	9	9	A few problems in *Genetic Programming III*	2
64-node Parsytec parallel machine	1995–2000	0.44	22	204	Most problems in *Genetic Programming III*	12
70-node Alpha parallel machine	1999–2001	3.2	7.3	1,481	A minority (8) of problems in *Genetic Programming IV*	2
1,000-node Pentium II parallel machine	2000–2002	30.0	9.4	13,900	A majority (28) of the problems in *Genetic Programming IV*	12

Table 5 is organized around the five just-explained order-of-magnitude increases in the expenditure of computing power. Column 4 of table 5 characterizes the qualitative nature of the results produced by genetic programming. Table 5 shows the progression of qualitatively more substantial results produced by genetic programming in terms of five order-of-magnitude increases in the expenditure of computational resources.

The order-of-magnitude increases in computing power shown in table 5 correspond closely (albeit not perfectly) with the following progression of qualitatively more substantial results produced by genetic programming:

- toy problems,
- human-competitive results not related to patented inventions,
- 20^{th}-century patented inventions,
- 21^{st}-century patented inventions, and
- patentable new inventions.

This progression demonstrates that genetic programming is able to take advantage of the exponentially increasing computational power made available by iterations of Moore's law. The progression of results shown in table 5 suggests that genetic programming may deliver increasingly more significant results in the future.

7 CONCLUSIONS

This paper demonstrated that genetic programming (1) now routinely delivers high-return human-competitive machine intelligence; (2) is an automated invention machine; (3) can automatically create a general solution to a problem in the form of a parameterized topology; and (4) has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time. These points were illustrated by a group of recent results involving the automatic synthesis of the topology and sizing of analog electrical circuits and involving the automatic synthesis of controllers.

Table 5 Progression of qualitatively more substantial results produced by genetic programming in relation to five order-of-magnitude increases in computational power

System	Period	Speed-up over previous row	Qualitative nature of the results produced by genetic programming
Serial Texas Instruments LISP machine	1987–1994	1 (base)	• Toy problems of the 1980s and early 1990s from the fields of artificial intelligence and machine learning
64-node Transtech transputer parallel machine	1994–1997	9	•Two human-competitive results involving one-dimensional discrete data (not patent-related)
64-node Parsytec parallel machine	1995–2000	22	• One human-competitive result involving two-dimensional discrete data • Numerous human-competitive results involving continuous signals analyzed in the frequency domain • Numerous human-competitive results involving 20^{th}-century patented inventions
70-node Alpha parallel machine	1999–2001	7.3	• One human-competitive result involving continuous signals analyzed in the time domain • Circuit synthesis extended from topology and sizing to include routing and placement (layout)
1,000-node Pentium II parallel machine	2000–2002	9.4	• Numerous human-competitive results involving continuous signals analyzed in the time domain • Numerous general solutions to problems in the form of parameterized topologies • Six human-competitive results duplicating the

			functionality of 21st-century patented inventions
Long (4-week) runs of 1,000-node Pentium II parallel machine	2002	9.3	• Generation of two patentable new inventions

REFERENCES

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

Aytur; Turgut Sefket. 2000. *Integrated Circuit with Variable Capacitor*. U. S. patent 6,013,958. Filed July 23, 1998. Issued January 11, 2000.

Cipriani, Stefano and Takeshian, Anthony A. 2000. *Compact cubic function generator*. U. S. patent 6,160,427. Filed September 4, 1998. Issued December 12, 2000.

Daun-Lindberg, Timothy Charles and Miller; Michael Lee. 2000. *Low Voltage High-Current Electronic Load*. U. S. patent 6,211,726. Filed June 28, 1999. Issued April 3, 2001.

Ikeuchi, Akira and Tokuda, Naoshi. 2000. *Voltage-Current Conversion Circuit*. U. S. patent 6,166,529. Filed February 24, 2000 in U. S.. Issued December 26, 2000 in U. S.. Filed March 10, 1999 in Japan.

Irvine, Robert and Kolb, Bernd. 2001. *Integrated Low-Pass Filter*. U.S. patent 6,225,859. Filed September 14, 1998. Issued May 1, 2001.

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

Koza, John R., and Rice, James P. 1992. *Genetic Programming: The Movie*. 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., 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., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido. 2003. *Genetic Programming IV. Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, Lanza, Guido, and Fletcher, David. 2003. *Genetic Programming IV Video: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Lee, Sang Gug. 2001. *Low Voltage Balun Circuit*. U. S. patent 6,265,908. Filed December 15, 1999. Issued July 24, 2001.

Samuel, Arthur L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development.* 3(3): 210–229.

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.

Ziegler, J. G. and Nichols, N. B. 1942. Optimum settings for automatic controllers. *Transactions of ASME*. (64)759–768.