Routine High-Return Human-Competitive Evolvable Hardware

John R. Koza
Stanford University
Stanford, California
koza@stanford.edu

Martin A. Keane
Econometrics Inc.
Chicago, Illinois
mak@sportsmrkt.com

Matthew J. Streeter
Genetic Programming Inc.
Mountain View, California
matts@cs.cmu.edu

Abstract

This paper reviews the use of genetic programming as an automated invention machine for the synthesis of both the topology and sizing of analog electrical circuits. The paper focuses on the importance of the developmental representation in this process. The paper makes the point that genetic programming now routinely delivers high-return human-competitive machine intelligence. It also makes the point that 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. The paper shows six examples where genetic programming has synthesized a circuit that duplicates the functionality or infringes a 21st-century patented electrical circuit. Finally, the paper discusses how genetic programming can be enhanced in order to potentially enable it to deliver more complex industrial-strength results.

1 Introduction

Genetic programming is an extension of the genetic algorithm (Holland 1975) into the arena of computer programs. 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 and analogs of recombination (crossover), mutation, gene duplication, gene deletion, and certain mechanisms of developmental biology to progressively breed an improved population over a series of many generations (Koza 1992; Koza 1994; Koza, Bennett, Andre, and Keane 1999; Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003; Banzhaf, Nordin, Keller, and Francone 1998; Langdon and Poli 2002).

Section 2 of this paper describes how genetic programming can be used to automatically synthesize the topology and sizing of analog electrical circuits and focuses on the importance of the developmental representation in this process.

Section 3 provides evidence that genetic programming now routinely delivers high-return human-competitive machine intelligence and evolvable hardware.

Section 4 discusses the progression of qualitatively more substantial results produced in synchrony with increasing computer power and asserts that genetic programming is able to take advantage of the exponentially increasing computational power made available by iterations of Moore's law.

Section 5 shows six human-competitive examples of evolvable hardware involving 21st-century patented inventions.

Section 6 discusses the commercial practicality of genetic programming for automated circuit synthesis, with emphasis on ways of improving the efficiency of runs to yield more complex industrial-strength results.

Section 7 is the conclusion.

2 The Importance of the Developmental Representation in the Automatic Synthesis of Circuits

The design process for electrical circuits begins with a high-level description of the circuit's desired behavior and characteristics. The process entails creation of both the topology and the sizing of a satisfactory circuit.

The *topology* of a circuit comprises

- the total number of components in the circuit,
- the type of each component (e.g., resistor, capacitor, transistor) at each location in the circuit,
- a list of the connections between the leads of the circuit's components, input ports, output ports, power sources, and ground.

The *sizing* of a circuit consists of the component value(s), if any, associated with each component. The sizing of a component is usually numerical.

Genetic programming was first used to automatically synthesize both the topology and sizing of analog electrical circuits in 1995 (Koza, Bennett, Andre, Keane 1996). Numerous examples of the automatic synthesis of analog electrical circuits composed of transistors, capacitors, resistors, inductors, and other components are found in Koza, Bennett, Andre, and Keane 1999 and Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003.

Our approach to the problem of automatically creating both the topology and sizing of an electrical circuit involves

(1) establishing a developmental representation for electrical circuits involving program trees, and

(2) defining a fitness measure that measures how well the behavior and characteristics of a candidate circuit satisfy the problem's high-level design requirements.

During the run, the evaluation of the fitness of each individual in the population involves

(1) converting each individual program tree in the population into a netlist for an electrical circuit,

(2) obtaining the circuit's behavior and characteristics, and

(3) using the circuit's behavior and characteristics to calculate fitness.

Electrical circuits are ordinarily represented as labeled graphical structures with cycles (circuit diagrams). However, the program trees used in genetic programming are *acyclic* graphs. Our approach to the automatic synthesis of circuits using genetic programming employs a developmental process to overcome this representational difference. This approach is inspired by the principles of developmental biology, the innovative work of Kitano (1990) on using developmental genetic algorithms to evolve neural networks, the creative and innovative work of Gruau (1992) on using developmental genetic programming (cellular encoding) to evolve neural networks, and early work on evolving Lindenmayer rules for creating structures (Koza 1993). The reader is also referred to ontogenetic programming (Spector and Stoffel 1996).

The developmental process transforms a program tree (an acyclic graph) into a fully developed electrical circuit (a graphical structure with cycles). The developmental process entails the execution of functions in a circuit-constructing program tree. The circuit-constructing program tree may contain component-creating functions, topology-modifying functions, development-controlling functions, arithmetic-performing functions, and automatically defined functions (ADFs).

The starting point for our developmental process consists of an initial circuit. The initial circuit consists of an embryo and a test fixture. The initial circuit is typically very simple. The embryo contains at least one modifiable wire. All development originates from the embryo's modifiable wire(s).

An electrical circuit is developed by progressively applying the functions in a circuit-constructing program tree to the modifiable wires of the original embryo and, as the circuit grows, to the modifiable wires and modifiable components that sprout from it. The execution of the functions in the program tree transforms the initial circuit into a fully developed circuit. That is, the functions in the circuit-constructing program tree progressively side-effect the embryo and its successor structures until a fully developed circuit eventually emerges.

A test fixture (external to the entity that is being automatically created) facilitates measurement of the performance and characteristics of the fully developed circuit. The test fixture is a hard-wired structure composed of nonmodifiable wires and nonmodifiable electrical components. The test fixture feeds external input(s) into the circuit that is being evaluated. It also enables the circuit's output(s) to be probed. The test fixture supplies the measurements that enable the fitness measure to assign a single numerical value of fitness to the behavior and characteristics of the fully developed circuit.

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

- component-creating functions that insert components (e.g., resistors, capacitors, transistors) into the developing circuit,
- topology-modifying functions (e.g., series division, parallel division, cut, via) that modify the topology of the developing circuit,
- development-controlling functions that control the developmental process by which the embryo and its successor structures are converted into a fully developed circuit (e.g., the development-ending function),
- arithmetic-performing functions (e.g., addition, subtraction) that may appear in a value-setting subtree that is an argument to a component-creating function and that specifies the numerical value of the component, and
- automatically defined functions (ADFs) that enable certain substructures to be reused (including parameterized reuse).

The component-creating functions generally have a value-setting subtree that establishes the value of the component (e.g., the capacitance of a capacitor).

Most of the component-creating and topology-modifying functions possess one or more construction-continuing subtrees.

The terminals in the circuit-constructing program trees may include

- constant numerical values,
- perturbable numerical values,
- externally supplied free variables,
- symbolic values (e.g., discrete alternative types for certain components), and
- zero-argument functions (e.g., the development-ending function, zero-argument automatically defined functions).

In a run of genetic programming, all the individual program trees created in generation 0 of the population are syntactically valid executable programs. All the genetic operations of genetic programming (i.e., crossover, mutation, reproduction, and the architecture-altering operations) operate so as to create syntactically valid executable programs from syntactically valid executable programs. Thus, all the individuals encountered during the run (including, in particular, the best-of-run individual) are syntactically valid executable programs.

Each circuit-constructing program tree is created in accordance with a constrained syntactic structure (strong typing) that imposes grammatical limits on how the available functions and terminals may be combined. For example, a value-setting subtree establishing the numerical value of a capacitor may only appear as a particular argument of the capacitor-creating function. All the individuals in the initial random population (generation 0) of a run of genetic programming for automatic circuit synthesis comply with the constrained syntactic structure. All the genetic operations that are performed during the run operate so as to preserve the constrained syntactic structure. Thus, all the individuals encountered during the run comply with the constrained syntactic structure.

The developmental approach is far more than just a mechanism for mapping an acyclic graph (the circuit-constructing program tree) into a graphical structure with cycles (the fully developed circuit).

For one thing, the developmental process has the advantage of preserving electrical connectivity. There are no unconnected leads in the initial circuit. Each component-creating, topology-modifying, and development-controlling function preserves electrical connectivity at each stage of the developmental process. Thus, there are no unconnected leads in the fully developed circuit.

More importantly, the developmental approach has the advantage of preserving locality. Most of the component-creating, topology-modifying, and development-controlling functions intentionally operate on a small local area of the circuit. Subtrees within the overall program tree therefore tend to represent a small local area. The crossover operation (the main workhorse of genetic programming and genetic algorithms) transplants subtrees. Thus, the crossover operation (in conjunction with the developmental process) tends to preserve locality. The mutation operation and architecture-altering operations similarly work in conjunction with the developmental process to preserve locality.

In addition to preserving locality, the developmental approach enables useful parts of a circuit-constructing program tree to be reused. Real-world circuits are replete with reuse. Reuse eliminates the need to "reinvent the wheel" on each occasion when a particular structure may be useful. Reuse makes it possible to exploit a problem's modularities, symmetries, parallelism, and regularities and thereby accelerate the problem-solving process. See Koza, Keane, and Streeter 2003 for a detailed discussion of the importance of reuse in automated circuit synthesis.

The efficiency of developmental genetic programming in the domain of automatic circuit synthesis stems from the combined effects of the

- preservation of syntactic validity,
- preservation of executablity of the circuit-constructing program trees,
- preservation of constrained syntactic structure,
- preservation of electrical connectivity,
- preservation of locality during crossover (and other operations), and
- the facilitation of reuse.

3 Genetic Programming Now Routinely Delivers High-Return Human-Competitive Machine Intelligence

We begin by defining what we mean by "human-competitive," "high-return," and "routine."

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. These problems typically have little relevance to any issues pursued by any scientist or engineer outside the fields of artificial intelligence and machine learning. To make the idea of human-competitiveness concrete, we say that a result is "human-competitive" if it satisfies one or more of eight criteria enumerated in Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003. One of the eight criteria (and the one most relevant to this paper) is

> "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."

In any event, all eight criteria have the desirable attribute of being at arms-length from the fields of artificial intelligence, machine learning, and genetic programming.

Table 1 Twenty-one previously patented inventions reinvented by genetic programming

Invention	Date	Inventor	Institution	Patent
Darlington emitter-follower section	1953	Sidney Darlington	Bell Telephone Laboratories	2,663,806
Ladder filter	1917	George Campbell	American Telephone and Telegraph	1,227,113
Crossover filter	1925	Otto Julius Zobel	American Telephone and Telegraph	1,538,964
"*M*-derived half section" filter	1925	Otto Julius Zobel	American Telephone and Telegraph	1,538,964
Cauer (elliptic) topology for filters	1934–1936	Wilhelm Cauer	University of Gottingen	1,958,742, 1,989,545
Sorting network	1962	Daniel G. O'Connor and Raymond J. Nelson	General Precision, Inc.	3,029,413
Computational circuits	NA	Numerous	Numerous	Numerous
Electronic thermometer	NA	Numerous	Numerous	Numerous
Voltage reference circuit	NA	Numerous	Numerous	Numerous
60 dB and 96 dB amplifiers	NA	Numerous	Numerous	Numerous
Second-derivative controller	1942	Harry Jones	Brown Instrument Company	2,282,726
Philbrick circuit	1956	George Philbrick	George A. Philbrick Researches	2,730,679
`NAND` circuit	1971	David H. Chung and Bill H. Terrell	Texas Instruments Incorporated	3,560,760
PID (proportional, integrative, and derivative) controller	1939	Albert Callender and Allan Stevenson	Imperial Chemical Limited	2,175,985
Negative feedback	1937	Harold S. Black	American Telephone and Telegraph	2,102,671
Low-voltage balun circuit	2001	Sang Gug Lee	Information and Communications University	6,265,908
Mixed analog-digital variable capacitor circuit	2000	Turgut Sefket Aytur	Lucent Technologies Inc.	6,013,958
High-current load circuit	2001	Timothy Daun-Lindberg and Michael Miller	International Business Machines Corporation	6,211,726
Voltage-current conversion circuit	2000	Akira Ikeuchi and Naoshi Tokuda	Mitsumi Electric Co., Ltd.	6,166,529
Cubic function generator	2000	Stefano Cipriani and Anthony A. Takeshian	Conexant Systems, Inc.	6,160,427
Tunable integrated active filter	2001	Robert Irvine and Bernd Kolb	Infineon Technologies AG	6,225,859

That is, a result cannot acquire the rating of "human-competitive" merely because it is considered interesting by researchers *inside* the specialized fields of artificial intelligence, machine learning, and genetic programming. 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 37 instances where genetic programming has produced a human-competitive result, of which 21 (table 1) are previously patented inventions of electrical circuits, networks, or controllers. In addition to these 21 instances, there are two instances where genetic programming has created a patentable new invention (Keane, Koza, and Streeter 2002) and there are 14 other instances of human-competitive results that are not patent-related, including the design of an X-Band Antenna for NASA's Space Technology 5 Mission (Lohn, Hornby, Kraus, Linden, Rodriguez, and Seufert 2003).

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.

Because each of the results in table 1 is a human-competitive result, 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 human-competitive 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 the 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 (i.e., the denominator) of the AI ratio.

The *executional steps* are the problem-independent and domain-independent steps that are automatically executed during a run of the problem-solving method. The *executional steps* of genetic programming include (1) generating the initial population of programs; (2) iteratively performing a main generational loop of executing each program, assigning a fitness value to each program, and creating the next generation of the population by applying genetic operations to program(s) selected from the population with a probability based on fitness, and (3) terminating the main generational loop and designating the individual with the best fitness as the result of the run. The result provides the "A" component (i.e., the numerator) of the AI ratio.

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) for each branch of the to-be-evolved computer program,

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

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

(4) various parameters for controlling the run, and

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

In practice, only a *de minimus* amount of "I" is contained in the primitive ingredients of the to-be-created computer program (the first and second preparatory steps), the problem's fitness measure (the third preparatory step containing the high-level statement of what needs to be done), and the run's 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, does not incorporate some method for specifying what needs to be done to guide the method's operation, does not employ parameters of some kind, and does not contain a termination criterion of some kind.

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 1 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.

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—that is, the fitness measure. Thus, after solving one problem of controller synthesis (say, the controller in table 1 patented by Callender and Stevenson in 1939 shown), the transition to each new problem of controller synthesis (say, the controller in table 1 patented by Jones in 1942) merely involves providing genetic programming with a different fitness measure. In other words, relatively little effort is required to make the required intra-domain transition.

Similarly, the vast majority of present-day electrical circuits on silicon chips are composed of transistors, capacitors, and resistors. Once the primitive ingredients are identified, new problems of circuit synthesis can be handled merely by changing the fitness measure.

In making the transition from problems involving, say, the automatic synthesis of controllers to problems involving, say, circuit synthesis, the primitive ingredients change from integrators, differentiators, gains, and the like to transistors, resistors, capacitors, and the like. The fitness measure changes from one involving, say, the controller's integral of time-weighted absolute error, overshoot, and disturbance rejection to a fitness measure that is based on, say, the circuit's amplification, suppression or passage of a signal, elimination of distortion, power supply rejection ratio, and the like. That is, relatively little effort is required to make an inter-domain transition.

Table 2 Human-competitive results produced by genetic programming with five computer systems

System	Period	Petacycles (10^{15}cycles) per day for system	Speed-up over previous row	Speed-up over first system in this table	Used for work in	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

4 Progression of Qualitatively More Substantial Results Produced in Synchrony with Increasing Computer Power

Numerous questions naturally arise in connection with any proposed approach to machine intelligence (including, specifically, genetic programming).

- Is the method formulated with sufficient precision to enable it to be implemented (or is it vagueware)?
- Has the method been successfully demonstrated on a specific single problem (or is it promiseware)?
- Has the method been applied to a difficult demonstrative problem (or is it toyware)?
  - Did the method top out after succeeding on a single demonstrative problem?
- Has the method solved multiple problems (or is it soloware)?
  - Are the multiple problems difficult?
  - Did the method top out at this stage?
- Has the method solved problems from multiple domains (or is it nicheware)?
  - Are the domains difficult?
  - Did the method top out at this stage?
- Were the results human-competitive—the bottom line of machine intelligence?
- Can the method profitably take advantage of the increased computational power available by means of parallel processing (or is it serialware)?
- Or, is the method Mooreware—able to take advantage of the exponentially increasing computational power made available by the relentless iteration of Moore's law?

Genetic Programming: On the Programming of Computers by Means of Natural Selection (Koza 1992a) demonstrated that genetic programming is not vagueware, promiseware, soloware, or nicheware.

The numerous human-competitive results produced by genetic programming (e.g., those in table 1) demonstrate that genetic programming is not toyware.

The final two questions in the above list address the issue of whether the proposed approach to machine intelligence has significant future potential.

Table 2 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 generated by each computer system.

The first entry in table 2 is a serial computer. The four subsequent entries are parallel computer systems. The presence of four increasingly powerful parallel computer systems in the table 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). In other words, genetic programming is not serialware.

Table 2 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 the table (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 (column 7).

An additional order-of-magnitude increase (beyond the four shown in table 2) was achieved by making extraordinarily long runs on the 1,000-node Pentium® II parallel machine. The length of the run that produced the genetically evolved controller (Keane, Koza, and Streeter 2002) 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. Counting this 9.3-to-1 increase as an additional speed-up, the overall speed-up between the first and last entries in the table is 130,660-to-1—five orders of magnitude.

Table 3 shows the progression of qualitatively more substantial results produced by genetic programming in terms of these five order-of-magnitude increases in computational resources:

- **Toy problems:** The LISP machine produced solutions to several dozen toy problems of the 1980s and early 1990s from the fields of artificial intelligence and machine learning.
- **Human-competitive results not related to patented inventions:** The 9-to-1 increase in computer power associated with the 64-node transputer parallel machine yielded two human-competitive results that were not patent-related.
- **20th-century patented inventions:** The 22-to-1 increase in computer power associated with the 64-node 80-MHz Parsytec parallel machine yielded numerous human-competitive results involving 20th-century patented inventions.
- **21st-century patented inventions:** The combined 69-to-1 increase in computer power associated with the next two computer systems (the 70-node 533-MHz Alpha parallel machine and 1,000-node 350-MHz Pentium II parallel machine) yielded numerous human-competitive results involving 21st-century patented inventions.
- **Patentable new inventions:** The 9-to-1 increase in computer power resulting from running the 1,000-Pentium II machine for 28.8 days yielded one of the controllers claimed as a new invention in a 2002 patent application (Keane, Koza, and Streeter 2002).

This progression demonstrates that genetic programming is able to take advantage of the exponentially increasing computational power made available by the relentless iteration of Moore's law. That is, genetic programming is Mooreware.

Table 3 Progression of qualitatively more substantial results produced by genetic programming

System	Period	Speed-up over previous	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 20th-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

5 Routine High-Return Human-Competitive Evolvable Hardware

There are now 21 instances where genetic programming has duplicated the functionality of a previously patented invention (including infringing a previously issued patent). Specifically, there are 15 instances where genetic programming has created an entity that either infringes or duplicates the functionality of a previously patented 20th-century invention and six instances where genetic programming has done the same with respect to a previously patented 21st-century invention.

To make the foregoing point concrete, this section presents the six post-2000 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, and resistors. The six inventions are the six inventions in table 1 that are dated 2000 or 2001. 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 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 (briefly 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 estimated elapsed time per generation.

Additional details are in Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003.

We now describe the six fitness measures.

5.1 Fitness Measures for the Six Problems

5.1.1 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.

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.5 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 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.

5.1.6 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 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.

5.2 Results for the Six Post-2000 Problems

5.2.1 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. The evolved circuit is superior to the patented circuit both in terms of its frequency response and its harmonic distortion.

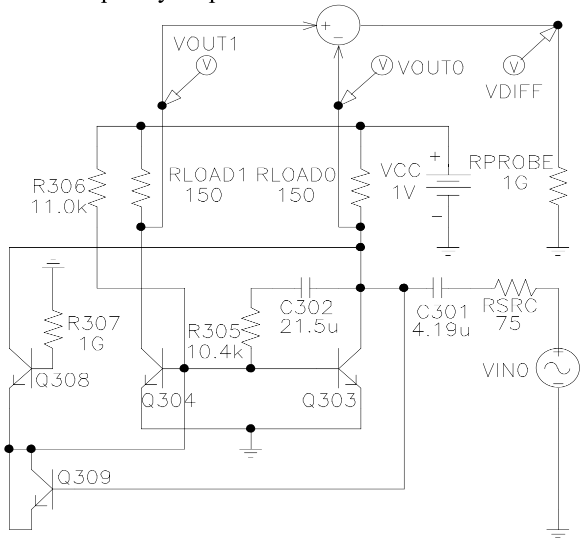

Figure 1 Best-of-run balun circuit

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

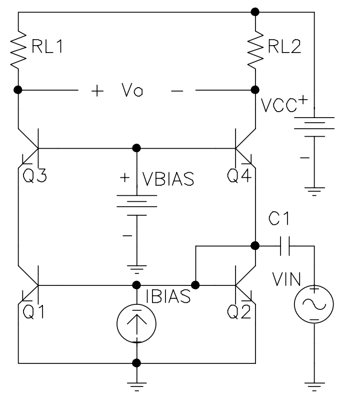

Figure 2 Prior art balun circuit shown in U.S. patent 6,265,908

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 that is called "C_{302}" in figure 1 and that is called "C_2" in figure 3.

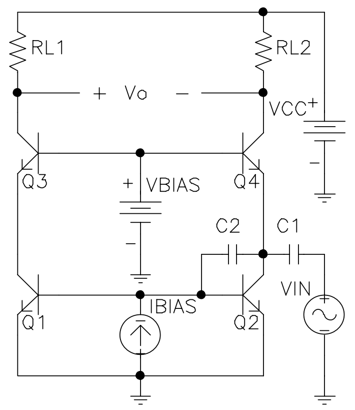

Figure 3 Lee's low voltage balun circuit shown in patent 6,265,908

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.

5.2.2 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).

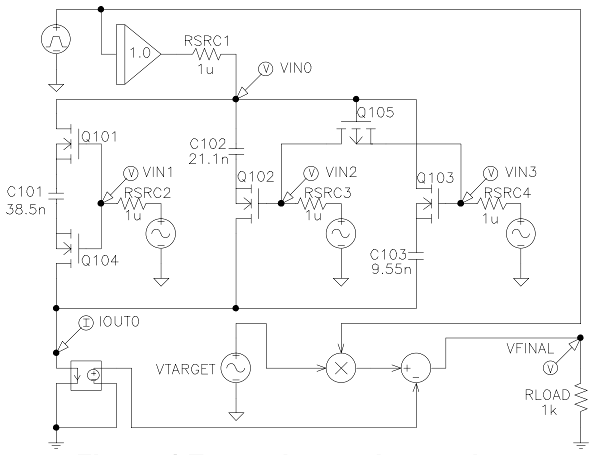

Figure 4 Evolved compliant register-controlled capacitor circuit

5.2.3 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 62% of the average 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.

5.2.4 High-Current Load Circuit

On generation 114, a circuit emerged that duplicated Daun-Lindberg and Miller's parallel FET transistor structure. The evolved circuit has 182% of the 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.

5.2.5 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.

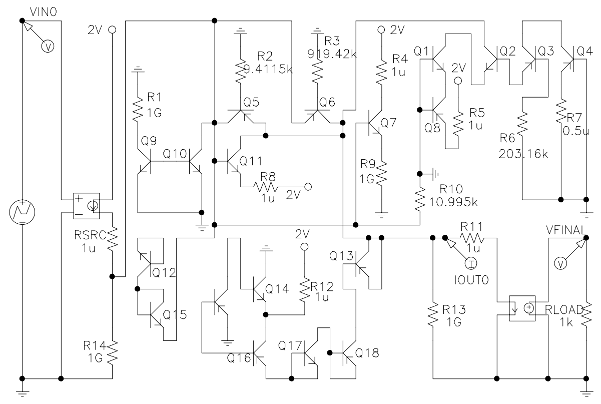

Figure 5 Best-of-run cubic signal generation circuit

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.

5.2.6 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.

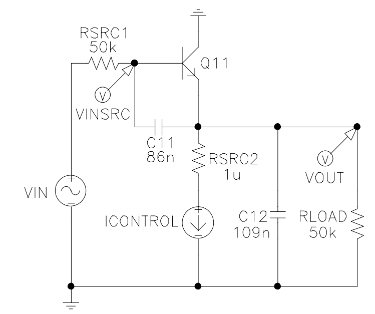

Figure 6 Best-of-run circuit for the tunable integrated active filter

6 Commercial Practicality of Genetic Programming for Automated Circuit Synthesis

The previous section demonstrates that genetic programming can automatically synthesize analog circuits that duplicate the functionality of six circuits that were patented after January 1, 2000.

Table 4 Computer time consumed by 11 runs of the six problems involving post-2000 patented inventions

Run	$M^*(i+1)$	Hours
Low-voltage balun circuit	490,000,000	25
Mixed analog-digital variable capacitor	198,000,000	88
High-current load circuit–1st run	230,000,000	134
High-current load circuit–2nd run	432,000,000	67
Voltage-current conversion circuit	550,000,000	83
Cubic function generator–1st run	915,000,000	206
Cubic function generator–2nd run	654,000,000	135
Tunable integrated active filter–1st run	142,000,000	23
Tunable integrated active filter–2nd run	102,000,000	14
Tunable integrated active filter–3rd run	78,000,000	12
Tunable integrated active filter–4th run	56,000,000	6

Table 4 tallies the computer time consumed by the 11 runs of the six post-2000 patented circuits. Column 2 of this table shows the product of the total population size, *M*, and the number of generations (*i*+1) run before the best-of-run individual was encountered. Column 3 shows the length of the run in hours.

As can be seen from table 4, the average number of hours for runs involving each of the six post-2000 patented circuits is 25, 88, 99, 83, 170, and 14, respectively. The average of these averages is 80 hours (3.3 days). (We use the average of the averages here because this table contains four runs of the problem that took the least computer time).

All six problems were run on a home-built parallel computer system consisting of 1,000 350-MHz Pentium II processors (appearing as the last row of table 2). This system operates at an overall rate of 3.5 $\times$ 10^{11} Hz. A 3.3-day (80-hour) run represents about 10^{17} cycles (i.e., 100 petacycles).

The relentless iteration of Moore's law promises increased availability of computational resources in future years. If available computer capacity continues to double approximately every 18 months over the next decade, a computation requiring 80 hours will require only about 1% as much computer time (i.e., about 48 minutes) a decade from now.

The question arises as to whether existing methods of genetic programming can be extended to deliver industrial-strength automated design of analog electrical circuits.

There are six promising factors suggesting that the previous results can be extended to deliver industrial-strength automated design of analog circuits and there are two countervailing factors that impede progress in that direction.

One promising factor is that multiple runs of a probabilistic algorithm are often necessary to solve a problem. We made 11 runs involving the post-2000 patented circuits (ignoring partial runs used for debugging purposes). All 11 runs produced a satisfactory solution. A success rate of 100% is unusual with a probabilistic algorithm. This high rate suggests that we are currently nowhere near the limit of the capability of the current techniques used to reinvent the six 21st-century patented circuits.

A second promising factor is that the previous work involving the six post-2000 patented circuits intentionally ignored numerous elementary and platitudinous pieces of domain knowledge about analog circuits. For example, previous runs did not cull egregiously flawed circuits, such as those drawing enormous amounts of current or those that were not connected to the circuit's incoming signals or output ports. Instead, the six problems were approached with a highly "clean hands" orientation—using as little problem-specific human-supplied domain knowledge about electrical circuits as possible. This "clean hands" orientation is, of course, entirely irrelevant to a practicing engineer interested in extending existing techniques to yield more complex industrial-strength results. The incorporation of such elementary and platitudinous domain knowledge thus creates considerable upside potential in the ability to automatically synthesize circuits.

A third promising factor is that the previous work intentionally ignored opportunities to employ elementary knowledge about the specific to-be-designed circuit. For example, the starting point for circuit development in previous runs consisted of a single modifiable wire and genetic programming was expected to automatically create the entire circuit from "nothing." However, a practicing engineer does not start each new assignment from first principles. Instead, the starting point is likely to incorporate one (and perhaps more) core subcircuits that are known to provide a good head start. For example, the search for a high-performance amplifier might begin with an embryo containing a balanced voltage gain stage and one or more modifiable wires as the starting point (as opposed to merely a single modifiable wire).

A fourth promising factor is that the previous work was intentionally uniform (and hence inefficient) in terms of genetic programming technique. For example, even when the problem had manifest parallelism, regularity, symmetry, and modularity, we intentionally did not permit the use of automatically defined functions (subroutines). However, a practicing engineer would recognize that reuse is highly relevant in at least two of the six problems involving the six post-2000 patented circuits in section 5 (namely the mixed analog-digital integrated circuit for variable capacitance and the low-voltage high-current transistor circuit for testing a voltage source). The benefits of using automatically defined functions in problems having parallelism, regularity, symmetry, and modularity are considerable (Koza 1990, Koza and Rice 1991, Koza 1992, Koza 1994). The removal of the previously enforced uniformity creates considerable additional upside potential in the ability to automatically synthesize circuits.

A fifth promising factor is that considerable work has been done in recent years to accelerate the convergence characteristics and general efficiency of circuit simulators. We used a version of the SPICE3 simulator (Quarles, Newton, Pederson, and Sangiovanni-Vincentelli 1994) that we modified in various ways (as described in Koza, Bennett, Andre, and Keane 1999). There are numerous commercially available simulators that are considerably more efficient than the version of the SPICE simulator that we used for the runs of the previous work involving the six post-2000 patented circuits. Speedups of up to 10-to-1 are reportedly possible today.

A sixth promising factor (already discussed in section 4) is that genetic programming has historically demonstrated the ability to profitably exploit the relentless increase in computer power suggested by Moore's law. The historical ability of genetic programming to yield progressively more substantial results with increased computer power suggests that even more substantial results will be possible in the future. Thus, the passage of time creates additional upside potential in the ability to automatically synthesize circuits.

There are, however, two countervailing factors that impede progress toward industrial-strength automated design of analog circuits.

The first countervailing factor concerns the nature of the multiobjective fitness measure that is typically associated with an industrial-strength problem. Previously published examples of the synthesis of analog circuits by means of genetic programming or genetic algorithms typically measure candidate circuits with a multiobjective fitness measure consisting of only a small number of different elements. For example, the fitness measure employed to synthesize the amplifier in chapter 45 of *Genetic Programming III: Darwinian Invention and Problem Solving* (Koza, Bennett, Andre, and Keane 1999) considered gain, bias, and distortion. The fitness measure employed to synthesize the amplifier in chapter 46 considered gain, bias, and distortion as well as the circuit's power supply rejection ratio (i.e., the circuit's ability to perform correctly in the face of fluctuations in the voltage provided by the circuit's external power supply). The fitness employed to synthesize and layout the amplifier in chapter 5 of *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003) considered gain, bias, distortion, as well as the area of the bounding rectangle after placement and routing on the substrate. In contrast, commercial circuits are described by detailed "data sheets" specifying the circuit's performance for a dozen or more characteristics. Each additional element in a fitness measure generally increases the computer time required to evaluate the candidate circuit. Moreover, as the number of elements in the fitness measure increases, the problem of efficiently combining the disparate elements ("apples and oranges") can become vexatious. In addition, previously published examples of the synthesis of analog circuits by means of genetic programming or genetic algorithms typically measure a candidate circuit with a single test fixture (test bench). However, the characteristics found in commercial data sheets are typically so different that they can only be measured by means of distinctly different test fixtures. Each different test fixture generally entails a different type of simulation (further increasing the total computer time required to fully evaluate the candidate circuit).

The second countervailing factor arises from the need to evaluate candidate circuits at the "corners" of various performance envelopes. For example, a real-world circuit might be required to operate correctly at –40° C and +105° C even though room temperature (27° C) may be the circuit's nominal ambient environment. Separate simulations (or, if reconfigurable hardware is being used, separate test scenarios) are required to measure the circuit's performance at each temperature—thus multiplying the required computer time by a factor of two (if only the two extremes are considered), three, or more. Similarly, a real-world circuit will be expected to operate correctly in the face of variation in the circuit's power supply (e.g., when the battery or other power supply is providing 4.5 volts or 5.5 volts, instead of a nominal 5.0 volts). Again, separate simulations are required to measure the circuit's performance at each voltage corner. In addition, a real-world circuit will be expected to operate correctly in the face of deviations between the behavior of an actual manufactured component and the component's "model" performance. Separate simulations may then be required, for example, for the component's "fast," "typical," and "slow" behavior. Circuits may also be expected to operate correctly in the face of variations in the load, variations in input characteristics, or variations in other characteristics. The combined effect of multiple independent sets of corners multiplies the required computer time by at least 2^N (where N is the number of sets of corners).

Thus, the answer to the question as to whether genetic programming can deliver industrial-strength automated design of analog electrical circuits depends on whether the six promising factors overcome the two countervailing factors.

In addition to the six promising factors mentioned above, runs of circuit synthesis problems can be accelerated in various ways.

First, many pieces of elementary knowledge helpful to the construction of useful circuits were not made available to the runs of the six problems involving 21st-century patented circuits. For example, the initial population of individuals in a run of genetic programming is typically created at random. As the run proceeds, new individuals are created by probabilistic problem-independent operations (e.g., crossover, mutation). Consequently, many individuals in the population represent unrealistic or impractical electrical circuits. One particularly egregious characteristic of the circuits that appear in unrestricted runs of genetic programming is that the circuit draws preposterously large amounts of current. To cull circuits of this type from the population, each circuit in the population can be examined for the current drawn by the circuit's positive power supply and negative power supply. If the current exceeds a certain generous maximum (e.g., an absolute value of 250 milliamperes), the circuit is penalized (or perhaps eliminated).

Second, a threshold requirement for a functioning circuit is that the circuit connect to all input signals, all output signals, and all necessary sources of power (e.g., the positive power supply and the negative power supply). A circuit can be easily tested for this characteristic and a high penalty value of fitness can be assigned to circuits failing this threshold test.

Third, the components that are inserted into a developing circuit need not be as primitive as a single transistor, resistor, or capacitor. Instead, the set of component-creating functions can be expanded to include numerous frequently-occurring, known-useful combinations of components. Examples include current mirrors, voltage gain stages, Darlington emitter-follower sections, cascodes, three-ported voltage divider subcircuits composed of two resistors in series, and three-ported subcircuits consisting of two resistors (or capacitors) with their common point connected to power or ground. In this vein, Graeb, Zizala, Eckmueller, and Antreich (2001) have identified (for a purpose entirely unrelated to evolutionary computation) a promising set of frequently-occurring combinations of transistors that are known to be useful in a broad range of analog circuits. For certain problems, the set of primitives can also be expanded to include higher-level entities, such as filters, op amps, oscillators, voltage-controller current sources, multipliers, and phase-locked loops.

Fourth, it is possible to integrate additional specific knowledge of electrical engineering into a run of genetic programming. For example, Sripramong and Toumazou (2002) combine current-flow analysis into their runs of genetic programming for the purpose of automatically synthesizing CMOS amplifiers.

Fifth, the authors believe that the efficiency of runs of genetic programming can, in general, be improved by adapting several of the principles set forth in *The Design of Innovation: Lessons from and for Competent Genetic Algorithms* (Goldberg 2002) to the domain of genetic programming.

Sixth, because there usually are multiple competing elements in the fitness measures of most practical problems of circuit synthesis, we believe that the efficiency of runs of genetic programming can be improved by using some of the recently published new techniques of multiobjective optimization (Deb 2001; Coello Coello, Van Veldhuizen, and Lamont 2002; and Zitzler, Deb, Thiele, Coello Coello, and Corne 2001).

Seventh, we believe that the efficiency of runs can be improved by adapting some of the innovative ideas in *Efficient and Accurate Parallel Genetic*

Algorithms (Cantu-Paz 2000) to the domain of genetic programming.

Eighth, there has been an outpouring of theoretical work in the past few years on the theory of genetic algorithms and genetic programming. In particular, the authors believe that many of the insights in *Foundations of Genetic Programming* (Langdon and Poli 2002) can be used to improve the efficiency of runs of genetic programming.

Looking forward, we believe that genetic programming will be increasingly used to automatically generate ever-more complex human-competitive results.

7 Conclusions

As far as we know, genetic programming is, at the present time, unique among methods of artificial intelligence and machine learning in terms of its duplication of numerous previously patented results, unique in its generation of patentable new results, unique in the breadth and depth of problems solved, and unique in its delivery of routine high-return, human-competitive evolvable hardware.

References

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

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

Cantu-Paz, Erick. 2000. *Efficient and Accurate Parallel Genetic Algorithms*. Boston: Kluwer Academic Publishers.

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.

Coello Coello, Carlos A., Van Veldhuizen, David A., and Lamont, Gary B. 2002. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Boston: Kluwer Academic Publishers.

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.

Deb, Kalyanmoy. 2001. *Multi-Objective Optimization using Evolutionary Algorithms*. Boston: Kluwer Academic Publishers.

Goldberg, David E. 2002. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Boston: Kluwer Academic Publishers.

Graeb, Helmut E., Zizala, S., Eckmueller, J., and Antreich, K. 2001. The sizing rules method for analog circuit design. *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. Piscataway, NJ: IEEE Press. Pages 343-349.

Gruau, Frederic. 1992a. Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In Schaffer, J. D. and Whitley, Darrell (editors). *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks 1992*. Los Alamitos, CA: The IEEE Computer Society Press.

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

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.

Keane, Martin A., Koza, John R., and Streeter, Matthew J. 2002. Automatic synthesis using genetic programming of an improved general-purpose controller for industrially representative plants. In Stoica, Adrian, Lohn, Jason, Katz, Rich, Keymeulen, Didier, and Zebulum, Ricardo (editors) 2002. *Proceedings of 2002 NASA/DoD Conference on Evolvable Hardware*. Los Alamitos, CA: IEEE Computer Society. Pages 113-122.

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

Koza, John R. 1990. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Stanford University Computer Science Department technical report STAN-CS-90-1314. June 1990.

Koza, John R., and Rice, James P. 1991. Genetic generation of both the weights and architecture for a neural network. In *Proceedings of International Joint Conference on Neural Networks*, *Seattle, July 1991*. Los Alamitos, CA: IEEE Press. Volume II. Pages 397-404.

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

Koza, John R. 1993. Discovery of rewrite rules in Lindenmayer systems and state transition rules in cellular automata via genetic programming. Symposium on Pattern Formation (SPF-93), Claremont, California. February 13, 1993.

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

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

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., Keane, Martin A., and Streeter, Matthew J. 2003. The importance of reuse and development in evolvable hardware. In Lohn, Jason, Zebulum, Ricardo, Steincamp, James, Keymeulen, Didier, Stoica, Adrian, and Ferguson, Michael I. (editors). 2003. *Proceedings of 2003 NASA/DoD Conference on Evolvable Hardware*. Los Alamitos, CA: IEEE Computer Society. Pages 33–42.

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.

Langdon, William B. and Poli, Riccardo. 2002. *Foundations of Genetic Programming*. Berlin: Springer-Verlag.

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

Lohn, Jason, Hornby, G., Kraus, W., Linden, Derek,. Rodriguez, A., and Seufert, S. 2003. Presentation at the 2003 NASA/DoD Conference on Evolvable Hardware entitled, "Evolutionary Design of an X-Band Antenna for NASA's Space Technology 5 Mission," Chicago, July, 2003.

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.

Sripramong, Thanwa and Toumazou, Christofer. 2002. The invention of CMOS amplifiers using genetic programming and current-flow analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 21(11). November 2002. Pages 1237–1252.

Spector, Lee and Stoffel, Kilian. 1996. Ontogenetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press. Pages 394–399.

Zitzler, Eckart, Deb, Kalyanmoy, Thiele, Lothar, Coello Coello, Carlos A., and Corne, David (editors). 2001. Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 2001, Proceedings. Lecture Notes in Computer Science. Volume 1993. Berlin, Germany: Springer-Verlag.