What's AI Done for Me Lately?

Genetic Programming's Human-Competitive Results

John R. Koza, *Stanford University*

Martin A. Keane, *Econometrics Inc.*

Matthew J. Streeter, *Genetic Programming Inc.*

The automated problem-solving technique of genetic programming has generated at least 36 human-competitive results. In six cases, it automatically duplicated the functionality of inventions patented after January 2000.

Producing human-competitive results is a primary reason why the AI and machine learning fields exist. As machine learning pioneer Arthur Samuel said in a 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."[1] So, what's AI done for us lately?

The answer—in terms of results produced by genetic programming's automatic synthesis of computer programs—is 36 human-competitive results, 21 of which duplicate previously patented inventions. (For more information on genetic programming, see the related sidebar.) We use the term *human-competitive* to mean that the result satisfies one or more of the following eight criteria:

1. 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.
2. 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.
3. 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.
4. The result is publishable in its own right as a new scientific result—independent of the fact that the result was mechanically created.
5. 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.
6. The result is equal to or better than a result that was considered an achievement in its field when it was first discovered.
7. The result solves a problem of indisputable difficulty in its field.
8. 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).

The criteria are at arms-length from AI and machine learning. A result cannot acquire the human-competitive rating merely because it interests researchers inside specialized fields attempting to create machine intelligence. Instead, the result must earn the rating independent of the fact that an automated method generated it.

Table 1 shows genetic programming's 36 human-competitive results, along with a criteria number (from the list just given) establishing the basis for the human-competitiveness claim. Here, we provide additional details on a sampling of these results.

Six patented inventions for analog electrical circuits

Patents represent current research and development efforts of the engineering and scientific communities. When an institution allocates time and money to

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

Table 1. Human-competitive results produced by genetic programming.

Claimed instance	Basis for claim (criteria number)
1. Creating a better-than-classical quantum-computing algorithm for the Deutsch-Jozsa "early promise" problem[2]	2, 6
2. Creating a better-than-classical quantum-computing algorithm for Grover's database search problem[3]	2, 6
3. Creating a quantum algorithm for the depth-two AND/OR query problem that is better than any previously published result[4,5]	4
4. Creating a quantum algorithm for the depth-one OR query problem that is better than any previously published result[5]	4
5. Creating a protocol for communicating information through a quantum gate that was previously thought not to permit such communication[6]	4
6. Creating a novel variant of quantum dense coding[6]	4
7. Creating a soccer-playing program that ranked in the middle of the field of 34 human-written programs in the RoboCup 1998 competition[7]	8
8. Creating four different algorithms for the transmembrane segment identification problem for proteins[8,9]	2, 5
9. Creating a sorting network for seven items using only 16 steps[9]	1, 4
10. Rediscovering the Campbell ladder topology for lowpass and highpass filters[9]	1, 6
11. Rediscovering the Zobel "*M*-derived half section" and "constant *K*" filter sections[9]	1, 6
12. Rediscovering the Cauer (elliptic) topology for filters[9]	1, 6
13. Automatically decomposing the problem of synthesizing a crossover filter[9]	1, 6
14. Rediscovering a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits[9]	1, 6
15. Synthesizing 60- and 96-decibel amplifiers[9]	1, 6
16. Synthesizing analog computational circuits for squaring, cubing, square root, cube root, logarithm, and Gaussian functions[9]	1, 4, 7
17. Synthesizing a real-time analog circuit for time-optimal control of a robot[9]	7
18. Synthesizing an electronic thermometer[9]	1, 7
19. Synthesizing a voltage reference circuit[9]	1, 7
20. Creating 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[9]	4, 5
21. Creating motifs that detect the D–E–A–D box family of proteins and the manganese superoxide dismutase family[9]	3
22. Synthesizing topology for a PID-D2 (proportional, integrative, derivative, and second derivative) controller[10]	1, 6
23. Synthesizing topology for a PID (proportional, integrative, and derivative) controller[10]	1, 6
24. Synthesizing an analog circuit equivalent to Philbrick circuit[10]	1, 6
25. Synthesizing a NAND circuit[10]	1, 6
26. Simultaneously synthesizing topology, sizing, placement, and routing of analog electrical circuits[10]	7
27. Rediscovering the Yagi-Uda antenna[10]	2, 6, 7
28. Creating PID tuning rules that outperform a PID controller using the Ziegler-Nichols and Åström-Hägglund tuning rules[10]	1, 2, 4, 5, 6, 7
29. Creating three non-PID controllers that outperform PID controllers using the Ziegler-Nichols and Åström-Hägglund tuning rules[10]	1, 2, 4, 5, 6, 7
30. Rediscovering negative feedback[10]	1, 6
31. Synthesizing a low-voltage balun circuit[10]	1
32. Synthesizing a mixed analog-digital variable capacitor circuit[10]	1
33. Synthesizing a high-current load circuit[10]	1
34. Synthesizing a voltage-current conversion circuit[10]	1
35. Synthesizing a cubic signal generator[10]	1
36. Synthesizing a tunable integrated active filter[10]	1

invent something and subsequently embarks on the time-consuming and expensive process of obtaining a patent, it deems the work of some scientific or practical importance. For this reason, we first describe a project in which we browsed the patent literature for patents on analog electrical circuits issued since January 2000 to commercial enterprises or university research institutions (see Table 2). We then used genetic programming to automatically synthesize both the structure (topology) and sizing (numerical component values) for circuits that duplicate the patented inventions' functionality. Only one of the six automatically created circuits infringes on the patent on which it is based.

Our method for automatically synthesizing analog circuits starts from a high-level statement of a circuit's desired behavior and characteristics and only minimal knowledge about analog electrical circuits. The method employs a circuit simulator for analyzing candidate circuits but does not rely on domain expertise or knowledge concerning the synthesis of circuits.

What is Genetic Programming?

Genetic programming is an automatic domain-independent method for solving problems. Starting with thousands of randomly created computer programs, it applies the Darwinian principle of natural selection, recombination (crossover), mutation, gene duplication, gene deletion, and certain mechanisms of developmental biology. It thus breeds an improved population over many generations.[1–4]

Genetic programming starts from a high-level statement of a problem's requirements and attempts to produce a computer program that solves the problem. The human user communicates the problem's high-level statement to the genetic programming system by performing certain preparatory steps. The five major preparatory steps for the basic version of genetic programming require the human user to specify

1. The set of terminals (for example, the problem's independent variables, zero-argument functions, and random constants) for each branch of the to-be-evolved program
2. The set of primitive functions for each branch of the to-be-evolved program
3. The fitness measure (for explicitly or implicitly measuring the fitness of individuals in the population)
4. Certain parameters for controlling the run
5. The termination criterion and the criterion for designating the run's result

The fitness measure is the primary mechanism for communicating the high-level statement of the problem's requirements to the genetic programming system. It specifies what needs to be done.

When using genetic programming to automatically synthesize computer programs, the programs are usually represented as rooted, point-labeled trees with ordered branches. The function set might consist of merely the ordinary arithmetic functions of addition, subtraction, multiplication, and division as well as a conditional branching operator.

When using genetic programming to automatically synthesize a specialized structure such as a controller, the function set consists of the specialized functions that make up controllers—integrators, differentiators, gains, adders, subtractors, leads, lags, and so forth.

When you use genetic programming to automatically synthesize electrical circuits, you must overcome an additional representational obstacle because circuits are labeled cyclic graphs. You can do so by establishing a mapping between the program trees that genetic programming ordinarily uses and the labeled cyclic graphs germane to circuits. You use a developmental process to map trees into circuits. This developmental process begins with a simple embryo (often consisting of just a single modifiable wire). You then develop an analog electrical circuit by progressively applying the circuit-constructing functions in a program tree to the embryo's initial modifiable wire (and to succeeding modifiable wires and modifiable components).

The functions in the circuit-constructing program trees include

- Topology-modifying functions that alter the topology of a developing circuit. The topology-modifying functions do such things as create a series or parallel division or create a connection from one point in a circuit to the power supply, to a distant point, or to ground.
- Component-creating functions that insert components (such as resistors, capacitors, and transistors) into a developing circuit.
- Development-controlling functions that control the developmental process (such as cut or end).

After the user has performed the preparatory steps, genetic programming executes a series of well-defined, problem-independent steps. Specifically, the genetic programming run starts by generating an initial population of compositions (typically random) of the problem's functions and terminals.

Then, genetic programming iteratively performs the following substeps (referred to herein as a generation) on the population of programs until satisfying the termination criterion.

First, execute each program in the population and assign it a fitness value using the problem's fitness measure.

Second, create a new population of programs by applying the following operations to selected individuals in the evolving population. The operations are applied to programs selected from the population with a probability based on fitness (with reselection allowed):

- *Reproduction*: Copy the selected program to the new population.
- *Crossover*: Create a new offspring program for the new population by recombining randomly chosen parts of two selected programs.
- *Mutation*: Create one new offspring program for the new population by randomly mutating a randomly chosen part of the selected program.
- *Architecture-altering operations*: Create one new offspring program for the new population by altering the program's architecture by creating, duplicating, or deleting a subroutine, iteration, loop, recursion, or element.

Finally, the individual with the best fitness run is designated as the result of the run. This result might solve (or approximately solve) the problem.

References

1. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, 1994.
2. J.R. Koza et al., *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, 1999.
3. J.R. Koza et al., *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Kluwer Academic Publishers, 2003.
4. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992

We read the patent document to determine the level of performance that the invention aimed to achieve. We then created a fitness measure reflecting the invention's performance and characteristics. This required translating the problem's high-level requirements into a precise computation. In particular, the fitness measure specified the desired time- or frequency-domain outputs, given various inputs. For each problem, a test fixture consisting of appropriate hard-wired components (such as a source resistor or load resistor) is connected to the input ports and desired output ports. The main difference between the genetic programming runs for

Table 2. Six post-2000 patented analog circuits.

Invention	Date	Inventor	Place	Patent
Low-voltage balun (balance/unbalance) circuit	2001	Sang Gug Lee	Information and Communications University	6,265,908
Mixed analog-digital circuit for variable capacitance	2000	Turgut Sefket Aytur	Lucent Technologies	6,013,958
Voltage-current conversion circuit	2000	Akira Ikeuchi and Naoshi Tokuda	Mitsumi Electric	6,166,529
Low-voltage high-current load circuit for testing a voltage source	2001	Timothy Daun-Lindberg and Michael Miller	International Business Machines	6,211,726
Low-voltage cubic function generator	2000	Stefano Cipriani and Anthony A. Takeshian	Conexant Systems	6,160,427
Tunable integrated active filter	2001	Robert Irvine and Bernd Kolb	Infineon Technologies	6,225,859

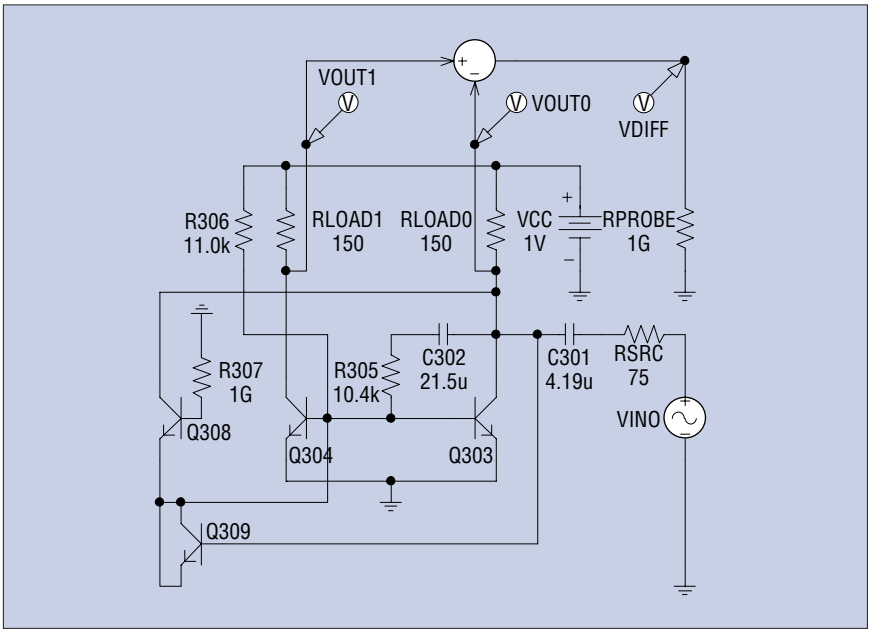


Figure 1. Genetically evolved low-voltage balun (balance/unbalance) circuit.

the six problems was that each had a different fitness measure.

Low-voltage balun circuit

A balun (balance/unbalance) circuit's purpose is to produce two outputs from a single input, each having half of the input's amplitude. One output should be in phase with the input while the other should be 180 degrees out of phase with the input, and both should have the same DC offset. The patented balun circuit uses a power supply of only 1 V (typical of low voltages that contemporary high-performance circuits demand).

We based the fitness measure for this problem on a frequency sweep analysis designed to measure the magnitude and phase of the circuit's two outputs and a Fourier analysis designed to measure harmonic distortion. The best-of-run evolved circuit (see Figure 1) is roughly a fourfold improvement over the patented circuit in terms of our fitness measure. The evolved circuit is superior both in terms of its frequency response and harmonic distortion.

The inventor states in the patent documents (US patent 6,265,908) that the essential difference between the prior art and his 2001 invention is a coupling capacitor C_2 located between the base and the collector of the transistor Q_2. Sang Gug Lee explains,

> The structure of the inventive balun circuit … is identical to that of [the prior art] except that a capacitor C_2 is 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.

(For this and any other patent mentioned, see http://patft.uspto.gov/netahtml/srchnum.htm and search by patent number.) This essential difference between the prior art and Lee's invention is an integral part of claim 1 of Lee's patent—a second capacitor C_2 coupled between the base and the collector of transistor Q_2.

The best-of-run genetically evolved circuit possesses the very feature (called C302 in Figure 1) that Lee identifies as the essence of his invention. (We discuss this in greater detail elsewhere.[10]) The genetically evolved circuit also matches three additional elements of claim 1 from Lee's patent. However, in spite of possessing the essence of Lee's invention, it does not match some other elements enumerated in claim 1 and thus does not infringe on the patent.

Mixed analog-digital register-controlled variable capacitor

The mixed analog-digital variable capacitor circuit has a capacitance controlled by the value stored in a digital register.

We based the fitness measure on the error accumulated by 16 combinations of time-domain test signals ranging over all eight possible values of a 3-bit digital register for two different analog input signals. The genetically evolved circuit performs as well as the patented circuit. It matches all but one of the elements of the patent's first claim (and hence does not infringe on the patent).

Tunable, integrated active filter

The tunable, integrated active filter's purpose is to perform the function of a lowpass filter whose passband boundary is dynamically specified by a control signal. We based the fitness measure on nine frequency-domain simulations, one for each of the passband boundary's nine values. We defined

error as the difference between an evolved circuit's frequency response and a model circuit's frequency response. We included parsimony in the fitness measure to get a small solution to the problem.

The genetically evolved circuit matches every element of claim 1 of US patent 6,225,859, thus infringing on it.

Voltage-current conversion circuit

The voltage-current conversion circuit's purpose is to take two voltages as input and to produce as output a stable current whose magnitude is proportional to the difference between the voltages. We based the fitness measure on four time-domain input signals. The genetically evolved circuit has roughly 62 percent of the average (weighted) error of the patented circuit (and outperformed the patented circuit on additional previously unseen test cases). The best-of-run circuit solves the problem in an entirely different manner from the patented circuit.

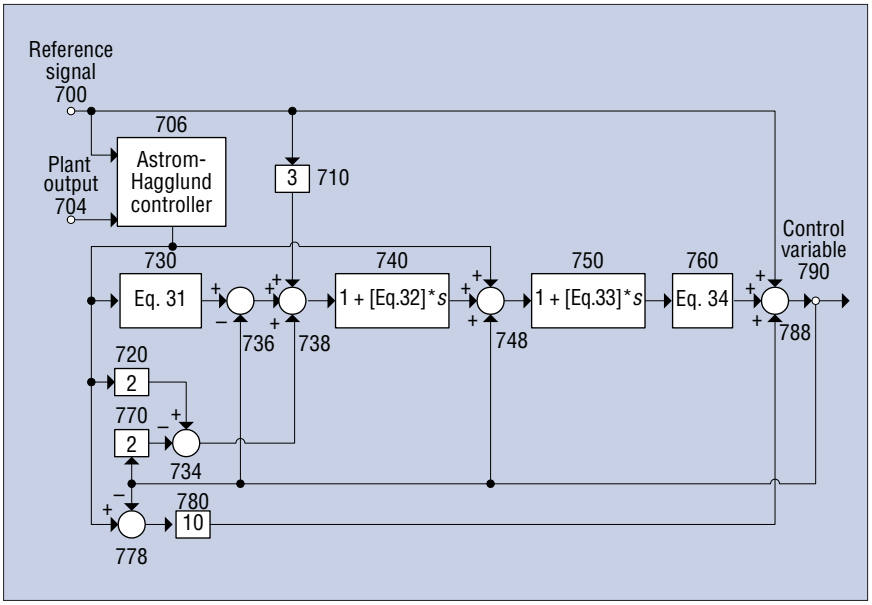


Figure 2. Genetically evolved improved non-PID parameterized controller.

High-current load circuit

US patent 6,211,726 covers a circuit designed to sink a time-varying amount of current in response to a control signal. The fitness measure consisted of two time-domain simulations, each representing a different control signal. The genetically evolved circuit shares the following features found in the patent's first claim:

> 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 the patent's first claim are very specific, and the genetically evolved circuit does not match these remaining elements. In fact, the genetically evolved circuit's remaining elements bear hardly any resemblance to the patented circuit. In this instance, genetic programming produced a circuit that duplicates the patented circuit's functionality using a different structure.

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. Providing only a 2-V power supply enforced the compactness constraint. The genetically evolved circuit has approximately 59 percent of the error of the patented circuit.

The claims in US patent 6,160,427 amount to a very specific description of the patented circuit. The genetically evolved circuit does not match these claims and, in fact, bears hardly any resemblance to the patented circuit.

Automatic synthesis of other complex structures

Genetic programming has also produced human-competitive results in the form of automatically synthesized controllers, antennas, classifier programs, and mathematical algorithms.

Controllers

Genetic programming created a controller for a two-lag plant containing proportional, integrative, derivative, and second derivative blocks. We were able to rediscover the PID-D2 (proportional-integral-derivative-second-derivative) topology covered by claim 38 of US patent 2,282,726 issued in 1942 to Harry Jones of the Brown Instrument Company of Philadelphia.[10]

Similarly, genetic programming created a controller containing a proportional, integrative, derivative block as described in claim 3 of US patent 2,175,985, issued in 1939 to Albert Callender and Allan Stevenson of Imperial Chemical Limited.[10]

Improved tuning rules for PID controllers. For the past six decades, most industrial users have relied on the tuning rules that John G. Ziegler and Nathaniel B. Nichols developed in 1942 to select the numerical parameters (for the gain of the controller's proportional, integrative, and derivative blocks and the reference signal's setpoint weighting) for the widely used PID type of controller. Karl J. Åström and Tore Hägglund improved upon these rules in 1995.

Recently, genetic programming synthesized tuning rules for PID controllers that outperform these existing rules.[10] We applied for a patent on these new improved tuning rules and the improved non-PID controller described next. If a patent is granted (as expected), we believe it will be the first one for an invention that genetic programming created.

Improved topology and tuning for general-purpose controllers. Genetic programming can automatically create, in a single run, a general (parameterized) solution to a problem in the form of a graphical structure whose edges represent components and where the parameter values of the components are specified by mathematical expressions containing free variables.

For example, genetic programming recently synthesized both the topology and sizing of an improved general-purpose non-PID controller (see Figure 2) that outperforms the

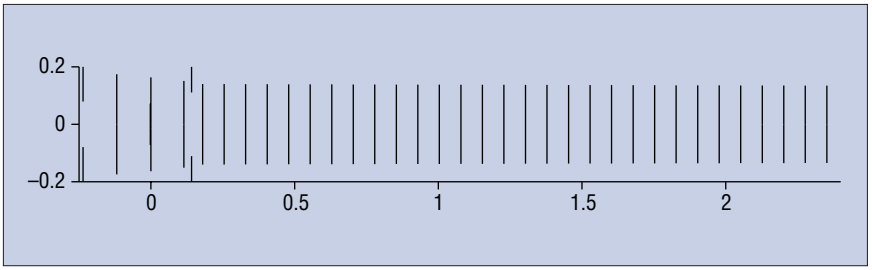


Figure 3. Genetically evolved Yagi-Uda antenna (measured in meters).

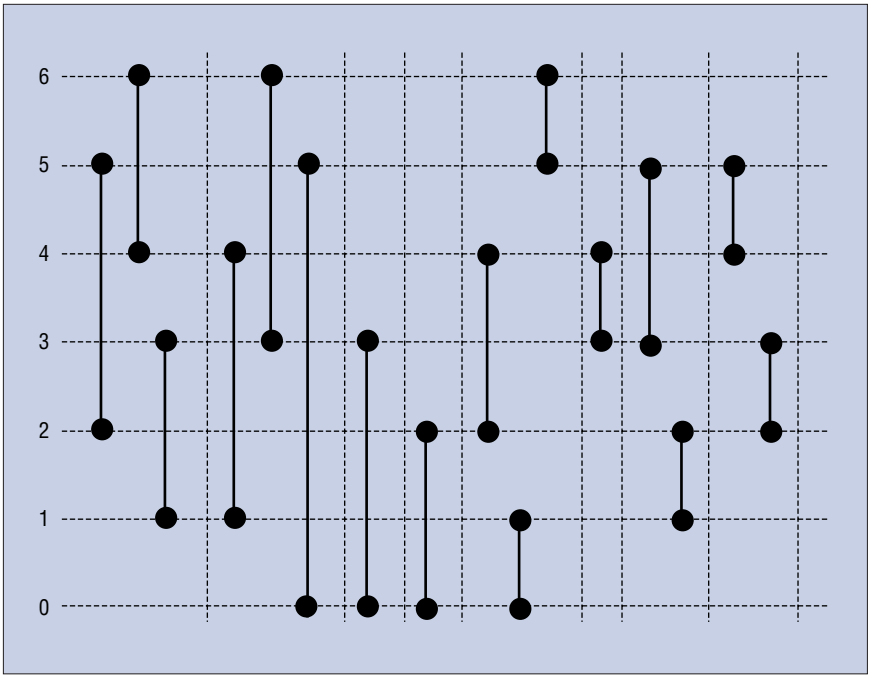


Figure 4. This 16-step minimal sorting network, evolved with Genetic Programming Problem Solver 2.0, is 100 percent correct.

general-purpose 1995 Åström-Hägglund PID controller and the widely used Ziegler-Nichols tuning rules.[10]

In the automated process, genetic programming determines the graph's size (its number of nodes) as well as its connectivity (specifying which nodes are connected to each other). Genetic programming also assigns component types (such as gain blocks, adders, subtractors, leads, lags, differentiators, and integrators) to the graph's nodes or edges. In addition, it creates mathematical expressions that establish the components' values, some of which contain free variables. The free variables let a single genetically evolved graphical structure represent a general (parameterized) solution to an entire category of problems. This genetically evolved controller is considered general-purpose because the parameter values for two gain blocks (730 and 760) and two lead blocks (740 and 750) are specified by mathematical expressions containing free variables for the plant's time constant T_r, dead time L, ultimate gain K_u, and ultimate period T_u. For example, gain block 730 in Figure 2 is parameterized by the genetically created equation

$$\left|\log\left|T_r - T_u + \log\left|\frac{\log\left(|L|^L\right)}{T_u + 1}\right|\right|\right|.$$

Antennas

To synthesize an antenna, a "turtle" deposits (or does not deposit) metal on a plane as it moves and turns under the control of various moving and turning functions (similar to those in the LOGO programming language). Genetic programming has rediscovered the Yagi-Uda antenna topology (see Figure 3) and synthesized an antenna that is competitive with a human-designed antenna for the same problem.[10]

A classifier program

Genetic programming has created a classifying program that identifies transmembrane domain proteins and has a lower error rate than the human-written algorithms for this problem.[9] For the genetically evolved classifier program, we did not prespecify

- That iterations should be used or, if used, the number of iterations
- That subroutines should be used or, if used, the number of subroutines and arguments possessed by each of them
- The precise number of steps in the result-producing branch, the subroutines, and the iteration-performing branches
- The exact sequence of steps performed in the result-producing branch, the subroutines, and the iteration-performing branches
- The hierarchical organization of the program's branches

All of the evolved solution's characteristics emerged during the genetic programming run as a result of using the architecture-altering operations for subroutines and iterations.

A sorting network

Genetic programming synthesized a 100 percent–correct 16-step sorting network (see Figure 4) that is superior (that is, having fewer compare-swap operations) to the one Daniel G. O'Connor and Raymond J. Nelson presented in their 1962 patent (US patent 3,029,413).

The Genetic Programming Problem Solver[9] uses a standardized set of functions and terminals, eliminating the need to prespecify a function set and terminal set for the problem. In addition, GPPS uses the architecture-altering operations to create, duplicate, and delete subroutines, loops, recursions, and internal storage during the genetic programming run. Because the evolving program's architecture is automatically determined during the run, GPPS eliminates the need to specify in advance whether to employ subroutines, loops, recursions, and internal storage in solving a given problem. It similarly eliminates

the need for the user to specify the number of arguments each subroutine possesses.

Assessing automated problem-solving techniques

We can assess an automated problem-solving technique in various ways. One way is to assess the gross amount of computer time that running the technique consumes.

Also, because genetic programming is a probabilistic algorithm (that is, it starts from a randomly created population, selects individuals to participate in the genetic operations probabilistically based on fitness, and executes some aspects of the genetic operations in a probabilistic way), we can measure the probability of a run's success under specified circumstances. This data makes it possible to calculate the computational effort[8] required to yield a solution to a given problem with a specified probability (such as 99 percent).

At a higher level, we can assess an automated problem-solving technique on the basis of whether it produces human-competitive results (as we did earlier).

Another way to assess an automated problem-solving technique concerns its routineness, which requires generality. More importantly, when we say that a method has a high degree of routineness, we mean 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 different domains. For example, the transition from controllers to antennas to classifying programs to sorting networks is accomplished by changing the function set from one specialized set of functions to another and appropriately changing the fitness measure to reflect the differing goals. This transition is routine with genetic programming.

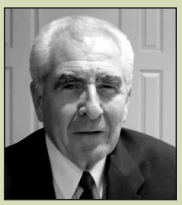
As computer time grows cheaper, researchers will routinely use genetic programming to produce useful new designs, generate patentable new inventions, and engineer around existing patents. ■

References

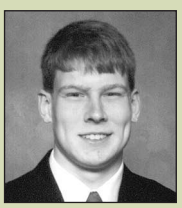
1. A.L. Samuel, "AI: Where It Has Been and Where It Is Going," *Proc. 8th Int'l Joint Conf. Artificial Intelligence*, Morgan Kaufmann, 1983, pp. 1152–1157.
2. L. Spector, H. Barnum, and H.L. Bernstein, "Genetic Programming for Quantum Computers," *Genetic Programming 1998: Proc. 3rd Ann. Conf.*, J.R. Koza et al., eds., Morgan Kaufmann, 1998, pp. 365–373.
3. L. Spector, H. Barnum, and H.J. Bernstein, "Quantum Computing Applications of Genetic Programming," *Advances in Genetic Programming 3*, L. Spector et al., eds., MIT Press, 1999, pp. 135–160.
4. L. Spector et al., "Finding a Better-than-Classical Quantum AND/OR Algorithm Using Genetic Programming," *Proc. 1999 Congress on Evolutionary Computation*, IEEE Press, 1999, pp. 2239–2246.
5. H. Barnum, H.J. Bernstein, and L. Spector, "Quantum Circuits for OR and AND of Ors," *J. Physics A: Mathematical and General*, vol. 33, no. 45, Nov. 2000, pp. 8047–8057.
6. L. Spector and H.J. Bernstein, "Communication Capacities of Some Quantum Gates, Discovered in Part through Genetic Programming," to be published in *Proc. 6th Int'l Conf. Quantum Comm., Measurement, and Computing*, Rinton Press, 2003.
7. D. Andre and A. Teller, "Evolving Team Darwin United," *RoboCup 98: Robot Soccer World Cup II*, LNCS 1604, M. Asada and H. Kitano, eds., Springer-Verlag, 1999, pp. 346–352.
8. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs,* MIT Press, 1994.
9. J.R. Koza et al., *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, 1999.
10. J.R. Koza et al., *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Kluwer Academic Publishers, 2003.

The Authors

John R. Koza is a consulting professor in the Biomedical Informatics Program in the Department of Medicine at Stanford University. He is also a consulting professor in the Department of Electrical Engineering at Stanford University, where he teaches a course on genetic algorithms and genetic programming. His research focuses on the creation of computer programs by automatic means. He received his PhD in computer science from the University of Michigan. He is a member of the IEEE. Contact him at Post Office Box K, Los Altos, CA 94023-4011; koza@stanford.edu.



Martin A. Keane is chief scientist of Econometrics Inc. of Chicago and a consultant to various computer-related and gaming-related companies. His research interests include applying genetic programming to the design of controllers for nonlinear dynamic systems. He received a PhD in mathematics from Northwestern University. He is a member of the IEEE. Contact him at 1960 N. Lincoln Park West, #1103, Chicago, IL 60614; martinkeane@ameritech.net.



Matthew J. Streeter is a systems programmer and researcher at Genetic Programming Inc. His primary research interest is applying genetic programming to problems of real-world scientific or practical importance. He received his MS in computer science from Worcester Polytechnic Institute. His masters thesis applied genetic programming to the automated discovery of numerical approximation formulae for functions and surfaces. He is a member of the IEEE. Contact him at 2585 Alvin Ave. #111, San Jose, CA 95121; matt@genetic-programming.com.