Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2008), *22*, 185–193. Printed in the USA.
Copyright © 2008 Cambridge University Press 0890-0604/08 $25.00
doi:10.1017/S0890060408000127

Human-competitive machine invention by means of genetic programming

JOHN R. KOZA
Biomedical Informatics Program, Department of Medicine, Stanford University, Stanford, California, USA

(Received June 21, 2007; Accepted July 18, 2007)

Abstract

Genetic programming is a systematic method for getting computers to automatically solve problems. 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, over a series of many generations, an improved population of candidate solutions to a problem. This paper makes the points that genetic programming now routinely delivers human-competitive machine intelligence for problems of automated design and can serve as an automated invention machine.

Keywords: Automated Design; Computer-Aided Design; Developmental Genetic Programming; Genetic Programming; Human-Competitive Results

1. INTRODUCTION

Design is a major activity of practicing engineers. Engineers are often called upon to design complex structures (e.g., electrical circuits, controllers, antennas, aerodynamic shapes, optical lens systems, and mechanical systems) that satisfy prespecified high-level design and performance goals. Some designs are sufficiently creative that they are considered to be inventions.

In designing complex structures, human engineers typically employ logic and previously known domain knowledge about the field of interest. Conventional approaches to automated design (such as those employing artificial intelligence) are typically knowledge intensive, logically sound, and deterministic.

Two of the most successful approaches to design, namely, the evolutionary process (occurring in nature) and the invention process (performed by creative humans), have characteristics that are significantly different from conventional approaches to automated design employing artificial intelligence. The evolutionary process in nature is not logical, deterministic, or knowledge intensive. The invention process (performed by creative humans) is not logical or deterministic. The fact that these two highly successful approaches to design are significantly different from conventional artificial intelligence approaches suggests that the evolutionary process and the invention process may contain significant lessons for the field automated design.

In nature, solutions to design problems are discovered by means of evolution and natural selection. Evolution is not deterministic. It does not rely on a knowledge base. In addition, evolution is certainly is not guided by mathematical logic. Indeed, one of the most important characteristics of the evolutionary process is that it actively generates and encourages the maintenance of inconsistent and contradictory alternatives throughout the entire duration of the process. Logically sound systems do not do that. They never entertain either inconsistency or contradictions. In contrast, the generation and maintenance of inconsistent and contradictory alternatives (called *genetic diversity*) is a known precondition for the success of the evolutionary process.

Likewise, the invention process (performed by creative humans) is a nondeterministic process, and it is not governed by logic. The invention process is typically characterized by a singular moment when the prevailing thinking concerning a longstanding problem is, in a "flash of genius," overthrown and replaced by a new approach that could not have been logically deduced from what was previously known. That is, inventions are characterized by a logical discontinuity that distinguishes the creative new design from that which can be logically deduced from what was previously known. In this connection, it is noteworthy that a purported invention is not considered to be worthy of a patent if the new idea can

Reprint requests to: John R. Koza, Biomedical Informatics Program, Department of Medicine, Stanford University, Stanford, CA 94305, USA. E-mail: koza@stanford.edu

be logically deduced from facts that are well known in a field by means of transformations that are well known in the field. A new idea is patentable only if there is an "illogical step," that is, a logically unjustified step. In the patent law, this legally required illogical step is sometimes referred to as a flash of genius, and it is the essence of inventiveness and creativity.

In short, both the invention process and the evolutionary process in nature and are very different from conventional artificial intelligence approaches to automated design.

Genetic programming is a method for getting computers to automatically solve problems. 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 is patterned after the evolutionary process in nature. Specifically, genetic programming employs 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, over a series of many generations, an improved population of candidate solutions to a problem. As will be demonstrated here, genetic programming can be employed to automatically synthesize designs for complex structures: it can be used to automate the design process. When genetic programming is so used, the process starts from a high-level specification of the structure's desired performance, which is the structure's characteristics and behavior. The outcome of a successful run of genetic programming is a structure that satisfies the user's prespecified performance requirements.

The automatic synthesis of the designs described in this paper is done *ab initio*, that is, without starting from a preexisting good design and without prespecifying the number of components in the structure being designed or the topological relationship among the components. The designs were created in a substantially similar and routine way, suggesting that the approach described in the paper can be readily applied to other similar design problems. The genetically evolved designs are instances of human-competitive results produced by genetic programming in the field of design.

Section 2 provides general background on genetic programming utilizing an automated design and invention technique patterned after the evolutionary process in nature.

Section 3 lists sources of additional information about genetic programming.

2. BACKGROUND ON GENETIC PROGRAMMING

The goal of getting computers to automatically solve problems is central to artificial intelligence, machine learning, and the broad area encompassed by what Turing called "machine intelligence" (Turing, 1948, 1950).

In his 1983 talk entitled "AI: Where It Has Been and Where It Is Going," machine learning pioneer Arthur Samuel (1983) stated the main goal of the fields of machine learning and artificial intelligence: "[T]he aim [is] to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence."

Genetic programming is a systematic method for getting computers to automatically solve a problem starting from a high-level statement of what needs to be done. Genetic programming is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. Genetic programming uses the Darwinian principle of natural selection along with analogs of recombination (crossover), mutation, gene duplication, gene deletion, and mechanisms of developmental biology to breed an ever-improving population of programs (Koza, 1989, 1990, 1992, 1994*a*, 1994*b*; Koza & Rice, 1992; Banzhaf et al., 1998; Koza, Bennett, Andre, & Keane, 1999; Koza, Bennett, Andre, Keane, & Brave, 1999; Koza et al., 2000; Koza, Keane, & Streeter, 2003; Koza, Keane, Streeter, Mydlowec, Yu, & Lanza, 2003; Koza, Keane, Streeter, Mydlowec, Yu, Lanza, & Fletcher, 2003).

Genetic programming is an extension of John Holland's genetic algorithm (Holland, 1975) to the arena of computer programs. Current concepts of genetic programming developed from the seminal work of numerous researchers in the 1970s and 1980s. Holland discussed the possibility of using the genetic algorithm to evolve sequences of assembly code. In 1978, Holland also proposed a broadcast language in which the genetic algorithm operated on structures more complex than fixed-length character strings. Holland and Reitman (1978) introduced the genetic classifier system in which sets of if–then logical production rules were evolved by means of a genetic algorithm. In 1980, Stephen F. Smith introduced the variable-length genetic algorithm and applied it to populations consisting of a hierarchy of if–then rules (Smith, 1980). In 1981, Forsyth introduced a highly innovative system called BEAGLE (Biological Evolutionary Algorithm Generating Logical Expressions) in which logical expressions were evolved in an evolutionary process (Forsyth, 1981). In the mid-1980s Nichael Cramer described highly innovative experiments in program induction employing Smith's crossover operation (Cramer, 1985); Hicklin (a student of John Dickinson at the University of Idaho) described a system with mutation and reproduction of programs (Hicklin, 1986); Fujiki (another student of Dickinson) applied all genetic operations to logical programs (Fujiki, l986); Fujiki and Dickinson (l987) performed induction of if–then clauses for playing the iterated prisoner's dilemma game; Antonisse and Keller (1987) applied genetic methods to higher level representations; and Bickel and Bickel (1987) applied genetic methods to if–then expert system rules.

Recent work on genetic programming is reported in the annual Genetic and Evolutionary Computation Conference (GECCO; Deb et al., 2004), the annual Euro-Genetic Programming conference (Keijzer et al., 2005), the annual Genetic Programming Theory and Applications workshop

(Yu et al., 2005), the annual Asia-Pacific Workshops on Genetic Programming (Cho et al., 2003), and in various other conferences and journals in the field of genetic and evolutionary computation, such as *Genetic Programming and Evolvable Hardware*. Additional sources of information about genetic programming, including links to available software for implementing genetic programming, can be found at www.genetic-programming.org.

2.1. Preparatory steps of genetic programming

Genetic programming starts from a high-level statement of the requirements of a problem and attempts to produce a computer program that solves the problem.

The human user communicates the high-level statement of the problem to the genetic programming algorithm by performing certain well-defined preparatory steps.

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

- the set of terminals (e.g., the independent variables of the problem, zero-argument functions, and random constants) for each branch of the program to be evolved,
- the set of primitive functions for each branch of the program to be evolved,
- the fitness measure (for explicitly or implicitly measuring the fitness of individuals in the population),
- certain parameters for controlling the run, and
- the termination criterion and method for designating the result of the run.

The first two preparatory steps specify the ingredients that are available to create the computer programs. A run of genetic programming is a competitive search among a diverse population of programs composed of the available functions and terminals.

The identification of the function set and terminal set for a particular problem (or category of problems) is usually a straightforward process. For some problems (such as symbolic regression), the function set may consist of merely the arithmetic functions of addition, subtraction, multiplication, and division as well as a conditional branching operator. The terminal set may consist of the program's external inputs (independent variables) and numerical constants.

For many other problems, the ingredients include specialized functions and terminals. For example, if the goal is to get genetic programming to automatically program a robot to mop the entire floor of an obstacle-filled room, the human user must tell genetic programming what the robot is capable of doing. For example, the robot may be capable of executing functions such as moving, turning, and swishing the mop. If the goal is the automatic creation of a controller, the function set may consist of integrators, differentiators, leads, lags, gains, adders, subtractors, and the like, and the terminal set may consist of signals such as the reference signal and plant output. If the goal is the automatic synthesis of an analog electrical circuit, it is necessary to provide ingredients that will enable genetic programming to construct circuits consisting of electrical components such as transistors, capacitors, and resistors, and to make connections among the components and the circuit's input and output ports.

The third preparatory step concerns the fitness measure for the problem. The fitness measure specifies what needs to be done. The fitness measure is the primary mechanism for communicating the high-level statement of the problem's requirements to the genetic programming system. For example, if the goal is to get genetic programming to automatically synthesize an amplifier, the fitness function is the mechanism for telling genetic programming to synthesize a circuit that amplifies an incoming signal (as opposed to, say, a filter circuit that suppresses the low frequencies of an incoming signal or a computational circuit that computes the square root of the incoming signal). The first two preparatory steps define the search space, whereas the fitness measure implicitly specifies the search's desired goal.

The fourth and fifth preparatory steps are administrative. The fourth preparatory step entails specifying the control parameters for the run. The most important control parameter is the population size. Other control parameters include the probabilities of performing the genetic operations, the maximum size for programs, and other details of the run.

The fifth preparatory step consists of specifying the termination criterion and the method of designating the result of the run. The termination criterion may include a maximum number of generations to be run as well as a problem-specific success predicate. The single best-so-far individual is usually then harvested and designated as the result of the run.

2.2. Executional steps of genetic programming

After the user has performed the preparatory steps for a problem, the run of genetic programming can be launched. Once the run is launched, a series of well-defined, problem-independent steps is executed.

Genetic programming typically starts with an initial population of randomly generated computer programs composed of the available programmatic ingredients (as provided by the human user in the first and second preparatory steps). These programs are typically generated by recursively generating a rooted point-labeled program tree composed of random choices of the primitive functions and terminals. The initial individuals are usually generated subject to a pre-established maximum size (specified by the user as a minor parameter in the fourth preparatory step). In general, the programs in the population are of different size (number of functions and terminals) and of different shape (the particular graphical arrangement of functions and terminals in the program tree).

Genetic programming iteratively transforms a population of computer programs into a new generation of the population by applying analogs of naturally occurring genetic operations. These operations are applied to individual(s) selected from the population. The individuals are probabilistically selected to participate in the genetic operations based on their fitness (as measured by the fitness measure provided by the human user in the third preparatory step). The iterative transformation of the population is executed inside the main loop (called a *generation*) of a run of genetic programming. The basic genetic operations are reproduction, crossover, and mutation. Architecture-altering operations and domain-specific operations are sometimes also used (although they are not used for the particular work described in this paper).

Specifically, genetic programming breeds computer programs to solve problems by executing the following three steps:

1. Generate an initial set (called the population) of compositions (typically random) of the functions and terminals appropriate for the problem.
2. Iteratively perform the following group of substeps (called a generation) on the population of programs until the termination criterion has been satisfied:
    a. Execute each program in the population and assign it a fitness value using the problem's fitness measure.
    b. Create a new population (the next generation) of programs by applying the following operations to program(s) selected from the population with a probability based on fitness (with reselection allowed).
        i. *Reproduction*: copy the selected program to the new population.
        ii. *Crossover*: create a new offspring program for the new population by recombining randomly chosen parts of two selected programs.
        iii. *Mutation*: create one new offspring program for the new population by randomly mutating a randomly chosen part of the selected program.
        iv. *Architecture-altering operations*: create one new offspring program for the new population by applying an operation to the selected program that alters the arrangement of the program's branches (e.g., subroutines, result-producing branches), the number of arguments possessed by each branch, and the nature of the hierarchical references allowed among the branches.
        v. *Domain-specific operations*: domain-specific operations are sometimes added to take advantage of design principles that are known to be helpful in a particular domain.
3. Designate an individual program (e.g., the individual with the best fitness) as the run's result.

This result may be a solution (or approximate solution) to the problem.

Genetic programming is problem independent in the sense that the above sequence of executional steps is not modified for each new run or each new problem. There is usually no discretionary human intervention or interaction during a run of genetic programming (although a human user often exercises judgment as to when to terminate a run).

2.3. Developmental genetic programming

The vast majority of the work applying genetic programming to design problems (including the work in this paper) employs a developmental process. When developmental genetic programming is used, the individuals that are genetically bred during the run of genetic programming are not themselves candidate structures in the domain involved. Instead, the individuals are computer programs consisting of instructions that transform a very simple initial structure (called the *embryo*) into a fully developed structure. For example, when developmental genetic programming is used to automatically design electrical circuits, the individuals in the population are not circuits but, instead, computer programs that specify how to construct a circuit, step by step, from some simple initial structure (often just a single wire).

The developmental representations used to apply genetic programming to design problems arise from early work in the field of genetic algorithms and genetic programming. In 1987 Wilson stated the following (Wilson, 1987, pp. 247–251):

> The genetic algorithm observes the genotype–phenotype distinction of biology: the algorithm's variation operators act on the genotype and its selection mechanisms apply to the phenotype. In biology, the genotype–phenotype difference is vast: the genotype is embodied in the chromosomes whereas the phenotype is the whole organism that expresses the chromosomal information. The complex decoding process that leads from one to the other is called biological development and is essential if the genotype is to be evaluated by the environment. Thus to apply the genetic algorithm to natural evolution calls for a representational scheme that both permits application of the algorithm's operators to the genotype and also defines how, based on the genotype, organisms are to be "grown," i.e., their development.

Kitano (1996) used a developmental process in conjunction with genetic algorithms to design neural networks using a graph generation system in 1990.

In 1992 Gruau described a technique in which genetic programming is used to concurrently evolve the architecture of a neural network, along with the weights, thresholds, and biases of each neuron in the neural network (Gruau, 1992*a*). In *Cellular Encoding of Genetic Neural Networks*, each individual program tree in the population of the run of genetic programming is a program for developing a complete neural network from a starting point consisting of a single embryonic neuron. In cellular encoding (sometimes also called "developmental genetic programming"), the developmental process for a neural network starts from an embryonic neural network consisting of a single neuron. The functions in the program tree specify how to develop the embryonic neural network into a full neural network. Certain functions permit a particular neuron to be subdivided in a parallel or sequential manner. Other functions can change the threshold of a neuron, the weight of a connection, or the bias of a neuron. Genetic programming is then used to breed populations of network-constructing program trees to evolve a neural network that is capable of solving particular problems. Gruau also described a version of his system using recursion (Gruau, 1992*b*, 1993, 1994*a*, 1994*b*; Gruau & Whitley, 1993). Whitley et al. (1995) applied developmental genetic programming to neurocontrol problems.

In 1993, Koza used genetic programming to evolve developmental rewrite rules (Lindenmayer system rules) using a "turtle" to create shapes such as the quadratic Koch island (Koza, 1993).

In 1994 Dellaert and Beer described "the synthesis of autonomous agents using evolutionary techniques" and presented "a simplified yet biologically defensible model of the developmental process" (Dellaert & Beer, 1994).

In 1994 Hemmi et al. noted, "Using a rewriting system, the system introduces a program development process that imitates the natural development process from the pollinated egg to adult and gives the HDL-program flexible evolvability" (Hemmi et al., 1994).

In 1994 Sims describes a system in which the morphological and behavioral components of virtual creatures are represented by directed graphs that evolve through the use of a graph-based genetic algorithm (Sims, 1994).

In 1996 Koza, Bennett, Andre, and Keane used developmental genetic programming to automatically synthesize a large body of analog electrical circuits, including several previously patented circuits (Koza et al., 1996*a*, 1996*b*, 1996*c*, 1996*d*). Circuit-constructing functions in the program tree specified how to develop a simple embryonic circuit (often containing just a single modifiable wire) into a fully developed circuit (containing transistors, capacitors, resistors, and other electronic components). Their method permitted distant connectivity within circuits by using vias. Koza and colleagues (1996*e*) provided for reuse of portions of circuits (by means of subroutines and iterations), parameterized reuse, and hierarchical reuse of substructures in evolving circuits.

In 1996 Brave used developmental genetic programming to evolve finite automata (Brave, 1996).

In 1996 Tunstel and Jamshidi used developmental methods for fuzzy logic systems (Tunstel & Jamshidi, 1996).

In 1996 Spector and Stoffel extended the notion of development to what they called "ontogenetic programming" (Spector & Stoffel, 1996*a*, 1996*b*).

> In nature, the structure and behavior of a mature organism is determined not only by its genetic endowment, but also by complex developmental processes that the organism undergoes while immersed in its environment (ontogeny). . . .
>
> Biologists refer to the developmental progression of an individual through its life span as ontogeny. In this paper we describe how rich ontogenetic components can be added to genetic programming systems, and we show how this can allow genetic programming to produce programs that solve more difficult problems. . . .
>
> Various morphological systems have been used in previous genetic programming systems to allow programs to "grow" into more complex forms prior to evaluation. Runtime memory mechanisms allow evolved programs to acquire information from their environments while they solve problems, and to change their future behavior on the basis of such information.
>
> Ontogenetic programming combines these ideas to allow for runtime modification of program structure. In particular, an ontogenetic programming system includes program self-modification functions in the genetic programming function set, thereby allowing evolved programs to modify themselves during the course of the run. . . .
>
> [W]e show how ontogenetic programming can be used to solve problems that would otherwise not be solvable. . . .
>
> We have shown that it is possible to use genetic programming to produce programs that themselves develop in significant, structural ways over the course of a run. We use the term "ontogenetic programming" to describe our technique for achieving this effect, which involves the inclusion of program self-modification functions in the genetic programming function set.

In 1996 Spector and Stoffel applied their methods to a symbolic regression problem (Spector & Stoffel, 1996*a*), a sequence prediction problem (Spector & Stoffel, 1996*a*), and a robotic agents problem (Spector & Stoffel, 1996*b*). They also describe how their methods can be used in conjunction with both tree and linear representations (Spector & Stoffel, 1996*b*).

In 1996 Luke and Spector described yet another variation on the developmental process (Luke & Spector, 1996):

> Like a cellular encoding, an edge encoding is a tree-structured chromosome whose phenotype is a directed graph, optionally with labels or functions associated with its edges and nodes. . . .
>
> Edge encoding, like cellular encoding, allows one to use standard S-expression-based Genetic Programming techniques to evolve arbitrary graph structures. The resulting graphs may be employed in various ways, for example as neural networks, as automata, or as knowledge-base queries. Each encoding scheme biases genetic search in a different way; for example, cellular encoding favors graphs with high edge/node ratios

while edge encoding favors graphs with low edge/node ratios. For this reason, we believe that certain domains will be much better served by one scheme than by the other.

2.4. Human-competitive results produced by genetic programming

Genetic programming can be applied to problems in a variety of fields, including design problems.

We say that a result produced by an automated method is "human-competitive" if it satisfies one of the following eight criteria (Koza et al., 2000; Koza, Keane, Streeter, Mydlowec, Yu, & Lanza, 2003):

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 at the time 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).

Focusing on patented inventions, there are at least 28 instances in which genetic programming has duplicated the functionality of a previously patented invention, infringed a previously issued patent, or created a patentable new invention (Koza, Keane, Streeter, Mydlowec, Yu, & Lanza, 2003). Specifically, there is one instance where genetic programming has created an entity that either infringes or duplicates the functionality of a previously patented 19th century invention, 15 instances where genetic programming has done the same with respect to a previously patented 20th century invention, 6 instances where genetic programming has done the same with respect to a previously patented 21st century invention, and 2 instances where genetic programming has created a patentable new invention (discussed in Section 6).

Table 1 provides information on these 28 human-competitive results that relate to previously patented inventions. Twelve of the results in Table 1 infringe previously issued patents and 10 duplicate the functionality of previously patented inventions in a noninfringing way.

It should also be mentioned that there are over a dozen additional known instances where genetic programming has produced a human-competitive result that are not patent related, including the design of an X-Band Antenna for NASA's Space Technology 5 Mission by Jason Lohn and his group at NASA Ames (Lohn et al., 2004), Lee Spector's quantum computing elements (Spector, Barnum, & Bernstein, 1998, 1999; Spector, Barnum, Bernstein, & Swamy, 1999; Spector, 2004), a sorting network (Koza, Bennett, Andre, & Keane, 1999), game-playing strategies (Luke, 1998; Andre & Teller, 1999), algorithms for cellular automata (Andre et al., 1996), and algorithms for protein segment classification (Koza, Bennett, Andre, & Keane, 1999). Starting in 2004, GECCO began giving awards for human-competitive results and many of these results were achieved by means of genetic programming (http://www.human-competitive.org).

3. ADDITIONAL SOURCES OF INFORMATION ABOUT GENETIC PROGRAMMING

In addition to the earlier citations, additional information about genetic programming may be obtained from books such as *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* (Langdon, 1998); *Automatic Re-engineering of Software Using Genetic Programming* (Ryan, 1999); *Data Mining Using Grammar Based Genetic Programming and Applications* (Wong & Leung, 2000); *Principia Evolvica: Simulierte Evolution mit Mathematica* (Jacob, 1997); *Illustrating Evolutionary Computation with Mathematica* (Jacob, 2001); *Genetic Programming* (Iba, 1996, in Japanese); *Evolutionary Program Induction of Binary Machine Code and Its Application* (Nordin, 1997); *Foundations of Genetic Programming* (Langdon & Poli, 2002); *Emergence, Evolution, Intelligence: Hydroinformatics* (Babovic, 1996); *Theory of Evolutionary Algorithms and Application to System Synthesis* (Blickle, 1997); and *Automatic Quantum Computer Programming: A Genetic Programming Approach* (Spector, 2004).

Additional information about genetic programming may be obtained from edited collections of papers such as the three *Advances in Genetic Programming* books from MIT Press (Kinnear, 1994; Angeline & Kinnear, 1996; Spector et al., 1999) and the proceedings of the Genetic Programming Conferences held between 1996 and 1998 (Koza, Goldberg, et al., 1996; Koza et al., 1997, 1998).

Table 1. *Twenty-eight previously patented inventions reinvented by genetic programming*

	Invention	Date	Inventor	Origin	Patent No.	Reference
1	Mechanical system composed of rigid members for drawing a straight line	1841	Robert Willis	Great Britain	G.B. 6258	Lipson (2004)
2	Ladder filter	1917	George Campbell	AT&T	U.S. 1,227,113	Section 25.15.1 of Koza, Bennett, Andre, & Keane (1999) and section 5.2 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
3	Crossover filter	1925	Otto Julius Zobel	AT&T	U.S. 1,538,964	Section 32.3 of Koza, Bennett, Andre, & Keane (1999)
4	“M-derived half section” filter	1925	Otto Julius Zobel	AT&T	U.S. 1,538,964	Section 25.15.2 of Koza, Bennett, Andre, & Keane (1999)
5	Cauer (elliptic) topology for filters	1934–1936	Wilhelm Cauer	University of Göttingen	U.S. 1,958,742 & U.S. 1,989,545	Section 27.3.7 of Koza, Bennett, Andre, & Keane (1999)
6	Negative feedback	1937	Harold S. Black	AT&T	U.S. 2,102,670 & U.S. 2,102,671	Chapter 14 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
7	PID controller	1939	Albert Callender & Allan Stervenson	Imperial Chemical Limited	U.S. 2,175,985	Section 9.2 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
8	Second-derivative controller	1942	Harry Jones	Brown Instrument Company	U.S. 2,282,726	Section 3.7 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
9	Darlington emitter–follower section	1953	Sidney Darlington	Bell Telephone Laboratories	U.S. 2,663,806	Section 42.3 of Koza, Bennett, Andre, & Keane (1999)
10	Philbrick circuit	1956	George Philbrick	George A. Philbrick Research	U.S. 2,730,679	Section 4.3 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
11	Sorting network	1962	Daniel G. O’Connor & Raymond J. Nelson	General Precision, Inc.	U.S. 3,029,413	Sections 21.4.4, 23.6, and 57.8.1 of Koza, Bennett, Andre, & Keane (1999)
12	NAND circuit	1971	David H. Chung & Bill H. Terrell	Texas Instruments Incorporated	U.S. 3,560,760	Section 4.4 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
13	Computation of circuits	Many	Many	Many	Many	Section 47.5.3 of Koza, Bennett, Andre, & Keane (1999)
14	Electronic thermometer	Many	Many	Many	Many	Section 49.3 of Koza, Bennett, Andre, & Keane (1999)
15	Voltage reference circuit	Many	Many	Many	Many	Section 50.3 of Koza, Bennett, Andre, & Keane (1999)
16	60- and 96-dB amplifiers	Many	Many	Many	Many	Section 45.3 of Koza, Bennett, Andre, & Keane (1999)
17	Cubic function generator	2000	Stefano Cipriani & Anthony A. Takeshian	Conexant Systems, Inc.	U.S. 6,160,427	Section 15.4.5 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
18	Mixed analog–digital variable capacitor circuit	2000	Turgut Sefket Aytur	Lucent Technologies Inc.	U.S. 6,013,958	Section 15.4.2 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
19	Voltage–current conversion circuit	2000	Akira Ikeuchi & Naoshi Tokuda	Mitsumi Electric Co., Ltd.	U.S. 6,166,529	Section 15.4.4 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
20	Low-voltage balun circuit	2001	Sang Gug Lee	Information and Communications University	U.S. 6,265,908	Section 15.4.1 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
21	High-current load circuit	2001	Timothy Daun-Lindberg & Michael Miller	IBM Corporation	U.S. 6,211,726	Section 15.4.3 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
22	Tunable integrated active filter	2001	Robert Irvine & Bernd Kolb	Infineon Technologies AG	U.S. 6,225,859	Section 15.4.6 of Koza, Keane, Streeter, Mydlowec, Yu, & Lanza (2003)
23	Telescope eyepiece	1940	Albert Konig	Carl Zeiss GmbH	U.S. 2,206,195	Koza et al., this issue
24	Telescope eyepiece system	1958	Robert B. Tackaberry & Robert M. Muller	American Optical Company	U.S. 2,829, 560	Koza et al., this issue
25	Eyepiece for optical instruments	1953	Maximillian Ludewig	Ernst Leitz GmbH	U.S. 2,637,245	Koza et al., this issue

Table 1. (*continued*)

	Invention	Date	Inventor	Origin	Patent No.	Reference
26	Wide angle eyepiece	1968	Wright H. Scidmore	U.S. Army	U.S. 3,390,935	Koza et al., this issue
27	Wide angle eyepiece	1985	Albert Nagler	No affiliation listed	U.S. 4,525,035	Koza et al., this issue
28	Telescope eyepiece	2000	Noboru Koizumi & Naomi Watanabe	Fuji Photo Optical Co., Ltd.	U.S. 6,069,750	Koza et al. (2005)

REFERENCES

Andre, D., Bennett III, F.H., & Koza, J.R. (1996). Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. *Genetic Programming 1996: Proc. 1st Annual Conf.* (Koza, J.R., Goldberg, D.E., Fogel, D.B., & Riolo, R.L., Eds.), pp. 3–11. Cambridge, MA: MIT Press.

Andre, D., & Teller, A. (1999). Evolving team Darwin United. In *RoboCup–98: Robot Soccer World Cup II, Lecture Notes in Computer Science* (Asada, M., & Kitano, H. Eds.), Vol. 1604, pp. 346–352. Berlin: Springer–Verlag.

Angeline, P.J., & Kinnear, Jr., K.E. (Eds.). 1996). *Advances in Genetic Programming 2*. Cambridge, MA: MIT Press.

Antonisse, H.J., & Keller, K. (1987). Genetic operators for high-level knowledge representations. *Proc. 2nd Int. Conf. Genetic Algorithms*. Mahwah, NJ: Erlbaum.

Babovic, V. (1996). *Emergence, Evolution, Intelligence: Hydroinformatics*. Rotterdam: Balkema.

Banzhaf, W., Nordin, P., Keller, R.E., & Francone, F.D. (1998). *Genetic Programming—An Introduction*. San Francisco, CA/Heidelberg: Morgan Kaufmann/dpunkt.

Bickel, A.S., & Bickel, R.W. (1987). Tree structured rules in genetic algorithms. *Proc. 2nd Int. Conf. Genetic Algorithms*. Mahwah, NJ: Erlbaum.

Blickle, T. (1997). *Theory of Evolutionary Algorithms and Application to System Synthesis*. TIK–Schriftenreihe Number 17. Zurich: vdf Hochschul Verlag AG an der ETH Zuerich.

Brave, S. (1996). Evolving deterministic finite automata using cellular encoding. *Genetic Programming 1996: Proc. 1st Annual Conf.* (Koza, J.R., Goldberg, D.E., Fogel, D.B., & Riolo, R.L., Eds.), pp. 39–44. Cambridge, MA: MIT Press.

Cho, S.-B., Nguyen, H.X., & Shan, Y. (Eds.). (2003). *Proc. 1st Asian-Pacific Workshop on Genetic Programming*. Accessed at www.aspgp.org

Cramer, N.L. (l985). A representation for the adaptive generation of simple sequential programs. *Proc. Int. Conf. Genetic Algorithms and Their Applications*. Mahwah, NJ: Erlbaum.

Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J., Harman, M., Holland, O., Lanzi, P.L., Spector, L., Tettamanzi, A., Thierens, D., & Tyrrell, A. (Eds.). (2004). *Genetic and Evolutionary Computation—GECCO 2004: Genetic and Evolutionary Computation Conf., Lecture Notes in Computer Science*, Vol. 3102. Berlin: Springer.

Dellaert, F., & Beer, R.D. (1994). Toward an evolvable model of development for autonomous agent synthesis. *Artificial Life IV: Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* (Brooks, R., & Maes, P., Eds.), pp. 246–257. Cambridge, MA: MIT Press.

Forsyth, R. (1981). BEAGLE—A Darwinian Approach to Pattern Recognition. *Kybernetes* 10, 159–166.

Fujiki, C. (l986). *An evaluation of Holland's genetic algorithm applied to a program generator*. MS Thesis. University of Idaho, Computer Science Department.

Fujiki, C., & Dickinson, J. (l987). Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma. *Proc. 2nd Int. Conf. Genetic Algorithms*. Mahwah, NJ: Erlbaum.

Gruau, F. (1992*a*). *Cellular Encoding of Genetic Neural Networks*. Technical Report 92-21. Ecole Normale Supérieure de Lyon, Laboratoire de l'Informatique du Parallélisme.

Gruau, F. (1992*b*). Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. *Proc. Workshop on Combinations of Genetic Algorithms and Neural Networks 1992* (Schaffer, J.D., & Whitley, D., Eds.). Los Alamitos, CA: IEEE Press.

Gruau, F. (1993). Genetic synthesis of modular neural networks. *Proc. 5th Int. Conf. Genetic Algorithms* (Forrest, S., Ed.), pp. 318–325. San Mateo, CA: Morgan Kaufmann.

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

Gruau, F. (1994*b*). Genetic micro programming of neural networks. In *Advances in Genetic Programming* (Kinnear, Jr., K.E. Ed.), pp. 495–518. Cambridge, MA: MIT Press.

Gruau, F., & Whitley, D. (1993). Adding learning to the cellular development process: a comparative study. *Evolutionary Computation 1*(*3*), 213–233.

Hemmi, H., Mizoguchi, J., & Shimohara, K. (1994). Development and evolution of hardware behaviors. *Artificial Life IV: Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* (Brooks, R., & Maes, P., Eds.), pp. 371–376. Cambridge, MA: MIT Press.

Hicklin, J.F. (1986). *Application of the genetic algorithm to automatic program generation*. MS Thesis. University of Idaho, Computer Science Department.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press.

Holland, J.H., & Reitman, J.S. (1978). Cognitive systems based on adaptive algorithms. In *Pattern-Directed Inference Systems* (Waterman, D.A., & Hayes-Roth, F., Eds.), pp. 313–329. New York: Academic.

Iba, H. (1996). *Genetic Programming*. Tokyo: Tokyo Denki University Press [in Japanese].

Jacob, C. (1997). *Principia Evolvica: Simulierte Evolution mit Mathematica*. Heidelberg: dpunkt.verlag.

Jacob, C. (2001). *Illustrating Evolutionary Computation with Mathematica*. San Francisco, CA: Morgan Kaufmann.

Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J., Tomassini, M. (Eds.). (2005). *Genetic Programming: 8th European Conf. (EuroGP 2005), Lecture Notes in Computer Science*, Vol. 3447. Heidelberg: Springer–Verlag.

Kinnear, Jr., K.E. (Ed.). (1994). *Advances in Genetic Programming*. Cambridge, MA: MIT Press.

Kitano, H. (1996). Morphogenesis for evolvable systems. In *Toward Evolvable Hardware, Lecture Notes in Computer Science* (Sanchez, E., & Tomassini, M., Eds.), Vol. 1062, pp. 99–117. Berlin: Springer–Verlag.

Koza, J.R. (1989). Hierarchical genetic algorithms operating on populations of computer programs. *Proc. 11th Int. Joint Conf. Artificial Intelligence*, Vol. 1, pp. 768–774. San Mateo, CA: Morgan Kaufmann.

Koza, J.R. (1990). *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Technical Report STAN-CS-90-1314, Stanford University, Computer Science Department.

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

Koza, J.R. (1993). Discovery of rewrite rules in Lindenmayer systems and state transition rules in cellular automata via genetic programming. *Symp. Pattern Formation (SPF–93)*, Claremont, CA, February 13.

Koza, J.R. (1994*a*). *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press.

Koza, J.R. (1994*b*). *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press.

Koza, J.R., Al-Sakran, S.H., & Jones, L.W. (2005). Automated re-invention of six patented optical lens systems using genetic programming. *Proc. Genetic and Evolutionary Computation Conf. (GECCO–2005)* (Beyer, H.-G., et al. Eds.), pp. 1953–1960. New York: ACM Press.

Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., & Riolo, R. (Eds.). (1998).

Genetic Programming 1998: Proc. 3rd Annual Conf. San Francisco, CA: Morgan Kaufmann.

Koza, J.R., Bennett III, F.H., Andre, D., & Keane, M.A. (1996*a*). Toward evolution of electronic animals using genetic programming. *Artificial Life V: Proc. 5th Int. Workshop on the Synthesis and Simulation of Living Systems* (Langton, C.G., & Shimohara, K., Eds.), pp. 327–334. Cambridge, MA: MIT Press.

Koza, J.R., Bennett III, F.H., Andre, D., & Keane, M.A. (1996*b*). Four problems for which a computer program evolved by genetic programming is competitive with human performance. *Proc. 1996 IEEE Int. Conf. Evolutionary Computation*, pp. 1–10. New York: IEEE Press.

Koza, J.R., Bennett III, F.H., Andre, D., & Keane, M.A. (1996*c*). Automated design of both the topology and sizing of analog electrical circuits using genetic programming. *Proc. Artificial Intelligence in Design '96* (Gero, J.S., & Sudweeks, F., Eds.), pp. 151–170. Dordrecht: Kluwer Academic.

Koza, J.R., Bennett III, F.H., Andre, D., & Keane, M.A. (1996*d*). Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming. *Genetic Programming 1996: Proc. First Annual Conf.* (Koza, J.R., Goldberg, D.E., Fogel, D.B., & Riolo, R.L., Eds.), pp. 123–131. Cambridge, MA: MIT Press.

Koza, J.R., Bennett III, F.H., Andre, D., & Keane, M.A. (1996*e*). Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. *Proc. Int. Conf. Evolvable Systems: From Biology to Hardware (ICES–96), Lecture Notes in Computer Science* (Higuchi, T., Iwata, M., & Liu, W., Eds.), Vol. 1259, pp. 312–326. Berlin: Springer–Verlag.

Koza, J.R., Bennett III, F.H., Andre, D., & Keane, M.A. (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann.

Koza, J.R., Bennett III, F.H., Andre, D., Keane, M.A., & Brave, S. (1999). *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann.

Koza, J.R., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M., Iba, H., & Riolo, R.L. (Eds.). (1997). *Genetic Programming 1997: Proc. 2nd Annual Conf.* San Francisco, CA: Morgan Kaufmann.

Koza, J.R., Goldberg, D.E., Fogel, D.B., & Riolo, R.L. (Eds.). (1996). *Genetic Programming 1996: Proc. 1st Annual Conf.* Cambridge, MA: MIT Press.

Koza, J.R., Keane, M.A., & Streeter, M.J. (2003). Evolving inventions. *Scientific America. 288(2)*, 52–59.

Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., & Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* New York: Kluwer Academic.

Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G., & Fletcher, D. (2003). *Genetic Programming IV Video: Routine Human-Competitive Machine Intelligence.* New York: Kluwer Academic.

Koza, J.R., Keane, M.A., Yu, J., Bennett III, F.H., & Mydlowec, W. (2000). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines 1*, 121–164.

Koza, J.R., & Rice, J.P. (1992). *Genetic Programming: The Movie.* Cambridge, MA: MIT Press.

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

Langdon, W.B., & Poli, R. (2002). *Foundations of Genetic Programming.* Berlin: Springer–Verlag.

Lipson, H. (2004). How to draw a straight line using a GP: benchmarking evolutionary design against 19th century kinematic synthesis. *Genetic and Evolutionary Conf. 2005 Late-Breaking Papers* (Keijzer, M., Ed.). Seattle, WA: International Society for Genetic and Evolutionary Computation.

Lohn, J.D., Hornby, G.S., & Linden, D.S. (2004). An evolved antenna for deployment on NASA's Space Technology 5 Mission. In *Genetic Programming Theory and Practice II* (O'Reilly, U.-M., Riolo, R.L., Yu, G., & Worzel, W., Eds.), Chap. 18. Boston: Kluwer Academic.

Luke, S. (1998). Genetic programming produced competitive soccer softbot teams for RoboCup97. *Genetic Programming 1998: Proc. Third Annual Conf.* (Koza, J.R., et al., Eds.), pp. 214–222. San Francisco, CA: Morgan Kaufmann.

Luke, S., & Spector, L. (1996). Evolving graphs and networks with edge encoding: Preliminary report. In *Late-Breaking Papers at the Genetic Programming 1996 Conf.* (Koza, J.R., Ed.), pp. 117–124. Stanford, CA: Stanford University Bookstore.

Nordin, P. (1997). *Evolutionary Program Induction of Binary Machine Code and its Application.* Munster, Germany: Krehl Verlag.

Ryan, C. (1999). *Automatic Re-engineering of Software Using Genetic Programming*. Amsterdam: Kluwer Academic.

Sims, K. (1994). Evolving 3D morphology and behavior by competition. *Artificial Life IV: Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* (Brooks, R., & Maes, P., Eds.), pp. 28–39. Cambridge, MA: MIT Press.

Smith, S.F. (1980). *A learning system based on genetic adaptive algorithms.* PhD Dissertation. University of Pittsburgh.

Spector, L. (2004). *Automatic Quantum Computer Programming: A Genetic Programming Approach.* Boston: Kluwer Academic.

Spector, L., Barnum, H., & Bernstein, H.J. (1998). Genetic programming for quantum computers. *Genetic Programming 1998: Proc. 3rd Annual Conf.* (Koza, J.R., et al., Eds.), pp. 365–373. San Francisco, CA: Morgan Kaufmann.

Spector, L., Barnum, H., & Bernstein, H.J. (1999). Quantum computing applications of genetic programming. In *Advances in Genetic Programming 3* (Spector, L., Langdon, W.B., O'Reilly, U.-M., & Angeline, P., Eds.), pp. 135–160. Cambridge, MA: MIT Press.

Spector, L., Barnum, H., Bernstein, H.J., & Swamy, N. (1999). Finding a better-than-classical quantum AND/OR algorithm using genetic programming. *IEEE Proc. 1999 Congr. Evolutionary Computation*, pp. 2239–2246. Piscataway, NJ: IEEE Press.

Spector, L., Langdon, W.B., O'Reilly, U.-M., & Angeline, P. (Eds.). (1999). *Advances in Genetic Programming 3.* Cambridge, MA: MIT Press.

Spector, L., & Stoffel, K. (1996*a*). Ontogenetic programming. 1996. *Genetic Programming 1996: Proc. 1st Annual Conf.* (Koza, J.R., Goldberg, D.E., Fogel, D.B., & Riolo, R.L., Eds.), pp. 394–399. Cambridge, MA: MIT Press.

Spector, L., & Stoffel, K. (1996*b*). Automatic generation of adaptive programs. *From Animals to Animats 4: Proc. 4th Int. Conf. Simulation of Adaptive Behavior* (Maes, P., Mataric, M.J., Meyer, J.-A., Pollack, J., & Wilson, S.W., Eds.), pp. 476–483. Cambridge, MA: MIT Press.

Tunstel, E., & Jamshidi, M. (1996). On genetic programming of fuzzy rule-based systems for intelligent control. *International Journal of Intelligent Automation and Soft Computing 2(3)*, 273–284.

Turing, A.M. (1948). Intelligent machinery. Reprinted in Ince, D.C. (Ed.). (1992). *Mechanical Intelligence: Collected Works of A. M. Turing*, pp. 107–127. Amsterdam: North Holland. Also reprinted in Meltzer B., & Michie D. (Eds.). (1969). *Machine Intelligence 5*. Edinburgh: Edinburgh University Press.

Turing, A.M. (1950). Computing machinery and intelligence. *Mind 59*(*236*), 433–460. Reprinted Ince D.C. (Ed.). (1992). *Mechanical Intelligence: Collected Works of A. M. Turing*, pp. 133–160. Amsterdam: North Holland.

Whitley, D., Gruau, F., & Preatt, L. (1995). Cellular encoding applied to neurocontrol. *Proc. 6th Int. Conf. Genetic Algorithms* (Eshelman, L.J., Ed.), pp. 460–467. San Francisco, CA: Morgan Kaufmann.

Wilson, S.W. (1987). The genetic algorithm and biological development. *Genetic Algorithms and Their Applications: Proc. 2nd Int. Conf. Genetic Algorithms* (Grefenstette, J.J., Ed.), pp. 247–251. Hillsdale, NJ: Erlbaum.

Wong, M.L., & Leung, K.S. (2000). *Data Mining Using Grammar Based Genetic Programming and Applications*. Amsterdam: Kluwer Academic.

Yu, G., Worzel, W., & Riolo, R. (Eds.). 2005. *Genetic Programming Theory and Practice III.* New York: Springer.

John R. Koza is a Consulting Professor in the Biomedical Informatics Program in the Department of Medicine and the Department of Electrical Engineering at Stanford University. He is the author of the 1992 book *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, three subsequent books on genetic programming, and over 250 papers on genetic programming, genetic algorithms, and artificial life.