1

Introduction

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).

Genetic programming is a systematic method for getting computers to automatically solve a problem. Genetic programming starts from a high-level statement of what needs to be done and automatically creates a computer program to solve the problem.

The most important point of this book is: Genetic programming now routinely delivers high-return human-competitive machine intelligence.

There are now 36 instances where genetic programming has produced a human-competitive result. In section 1.1, we define "routine," "high-return," "human-competitive," and "machine intelligence" and outline the evidence supporting each claimed human-competitive result.

The second of this book's four main points is: Genetic programming is an automated invention machine.

There are now 23 instances where genetic programming has duplicated the functionality of a previously patented invention, infringed a previously issued patent, or created a patentable new invention. Specifically, there are 15 instances where genetic programming has created an entity that either infringes or duplicates the functionality of

Table 1.1 Four main points of this book

	Main point
1	• Genetic programming now routinely delivers high-return human-competitive machine intelligence.
2	• Genetic programming is an automated invention machine.
3	• Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology.
4	• Genetic programming has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time.

a previously patented 20th-century invention, six instances where genetic programming has done the same with respect to an invention patented after January 1, 2000, and two instances where genetic programming has created a patentable new invention. The two new inventions are general-purpose controllers that outperform controllers employing tuning rules that have been in widespread use in industry for most of the 20th century.

Novelty and creativity are prerequisites for patentability. A new idea that can be logically deduced from facts that are known in a field, using transformations that are known in a field, is not considered to be patentable by the Patent Office. A new idea is patentable only if there is an "illogical step" (that is, a logically unjustified step) that distinguishes the proposed invention from that which is readily deducible from what is already known. As we discuss in section 1.2, genetic programming often unearths novel solutions to problems because it does not travel along the well-trod paths of previous human thinking. The inventions generated by genetic programming exhibit the kind of illogical discontinuity from previous human work that is required to obtain a patent.

The third main point of this book is: Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology.

Eleven problems in this book demonstrate that genetic programming can automatically create, in a single run, a general (parameterized) solution to a problem in the form of a graphical structure whose nodes or edges represent components and where the parameter values of the components are specified by mathematical expressions containing free variables. Section 1.3 previews the automatic creation of such parameterized topologies.

This book's fourth main point is: Genetic programming has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time.

Section 1.4 discusses the progression of results produced by genetic programming over the 15-year period from 1987 to 2002, including

- solving toy problems,
- producing human-competitive results not involving previously patented inventions,
- duplicating 20th-century patented inventions,
- duplicating 21st-century patented inventions, and
- creating patentable new inventions.

Table 1.1 shows the four main points of this book.

In addition to solving numerous problems involving analog electrical circuits (chapters 4, 5, 10, 11, 14, and 15) and controllers (chapters 3, 9, 12, and 13), the book presents results involving the automatic synthesis of networks of chemical reactions (chapter 8), antennas (chapter 6), and genetic networks (chapter 7).

Chapter 2 provides general background on genetic programming. Chapters 9, 10, 11, and 13 discuss parameterized topologies. Chapter 16 discusses the characteristics that may make certain problems better suited for genetic algorithms or genetic programming. Chapter 17 discusses issues of parallelization and computer time. Chapter 18 provides a historical perspective on computer speed and the succession of qualitatively more substantial results produced by genetic programming.

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, unique in its demonstrated ability to produce parameterized topologies, and unique in its delivery of routine high-return, human-competitive machine intelligence.

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

Focusing on this book's first main point (i.e., that genetic programming now routinely delivers high-return human-competitive machine intelligence), the next four subsections explain what we mean by the terms

- human-competitive (section 1.1.1),
- high-return (section 1.1.2),
- routine (section 1.1.3), and
- machine intelligence (section 1.1.4).

Then, four additional sub-sections outline the evidence that supports the claim that genetic programming now delivers results with these four characteristics.

1.1.1 What We Mean by "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.

In his 1983 talk entitled "AI: Where It Has Been and Where It Is Going," machine learning pioneer Arthur Samuel said:

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

Samuel's statement reflects the common goal articulated by the pioneers of the 1950s in the fields of artificial intelligence and machine learning. Indeed, getting machines to produce human-like results is *the* reason for the existence of the fields of artificial intelligence and machine learning.

To make this goal more concrete, we say that a result is "human-competitive" if it satisfies one or more of the eight criteria in table 1.2.

The eight criteria in table 1.2 have the desirable attribute of being at arms-length from the fields of artificial intelligence, machine learning, and genetic programming. That is, a result cannot acquire the rating of "human-competitive" merely because it is endorsed by researchers *inside* the specialized fields that are attempting to create

Table 1.2 Eight criteria for saying that an automatically created result is human-competitive

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

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

These eight criteria are the same as those presented in *Genetic Programming III: Darwinian Invention and Problem Solving* (Koza, Bennett, Andre, and Keane 1999a).

1.1.2 What We Mean by "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.

Deep Blue: An Artificial Intelligence Milestone (Newborn 2002) describes the 1997 defeat of the human world chess champion Garry Kasparov by the Deep Blue computer system. This outstanding example of machine intelligence is clearly a human-competitive result (by virtue of satisfying criterion H of table 1.2). Feng-Hsiung Hsu (the system architect and chip designer for the Deep Blue project) recounts the intensive work on the Deep Blue project at IBM's T. J. Watson Research Center between 1989 and 1997 (Hsu 2002). The team of scientists and engineers spent years developing the software and the specialized computer chips to efficiently evaluate large numbers of alternative moves as part of a massive parallel state-space search. In short, the human developers invested an enormous amount of "I" in the

project. In spite of the fact that Deep Blue delivered a high (human-competitive) amount of "A," the project has a low return when measured in terms of the A-to-I ratio. The builders of Deep Blue convincingly demonstrated the high level of intelligence of the humans involved in the project, but very little in the way of machine intelligence.

The Chinook checker-playing computer program is another impressive human-competitive result. Jonathan Schaeffer recounts the development of Chinook by his eight-member team at the University of Alberta between 1989 and 1996 in his book *One Jump Ahead: Challenging Human Supremacy in Checkers* (Schaeffer 1997). Schaeffer's team began with analysis. They recognized that the problem could be profitably decomposed into three distinct subproblems. First, an opening book controls the play at the beginning of each game. Second, an evaluation function controls the play during the middle of the game. Finally, when only a small number of pieces are left on the board, an endgame database takes over and dictates the best line of play. Perfecting the opening book entailed an iterative process of identifying "positions where Chinook had problems finding the right move" and looking for "the elusive cooks" (Schaeffer 1997, page 237). By the time the project ended, the opening book had over 40,000 entries. In a chapter entitled "A Wake-Up Call," Schaeffer refers to the repeated difficulties surrounding the evaluation function by saying "the thought of rewriting the evaluation routine … and tuning it seemed like my worst nightmare come true." Meanwhile, the endgame database was painstakingly extended from five, to six, to seven, and eventually eight pieces using a variety of clever techniques. As Schaeffer (page 453) observes,

> "The significant improvements to Chinook came from the knowledge added to the program: endgame databases (computer generated), opening book (human generated but computer refined), and the evaluation function (human generated and tuned). We, too, painfully suffered from the knowledge-acquisition bottleneck of artificial intelligence. Regrettably, our project offered no new insights into this difficult problem, other than to reemphasize how serious a problem it really is."

Chinook defeated world champion Marion Tinsley. However, because of the enormous amount of human "I" invested in the project, Chinook (like Deep Blue) has a low return when measured in terms of the A-to-I ratio.

The aim of the fields of artificial intelligence and machine learning is to get computers to automatically generate human-competitive results with a high AI ratio—not to have humans generate human-competitive results themselves.

1.1.3 What We Mean by "Routine"

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

What fraction of Deep Blue's and Chinook's highly specialized software, hardware, databases, and evaluation techniques can be brought to bear on different games? For example, can Deep Blue's massive parallel state-space search or Chinook's three-way decomposition be gainfully applied to a game, such as Go, with

a significantly larger number of possible alternative moves at each point in the game? What fraction of these systems can be applied to a game of incomplete information, such as bridge? What more broadly applicable principles are embodied in these two systems? For example, what fraction of these methodologies can be applied to the problem of getting a robot to mop the floor of an obstacle-laden room? Correctly recognizing images or patterns? Devising an algorithm to solve a mathematical problem? Automatically synthesizing a complex structure?

A problem-solving method cannot be considered routine if its executional steps must be substantially augmented, deleted, rearranged, reworked, or customized by the human user for each new problem.

1.1.4 What We Mean by "Machine Intelligence"

We use the term "machine intelligence" to refer to the broad vision articulated in Alan Turing's 1948 paper entitled "Intelligent Machinery" and his 1950 paper entitled "Computing Machinery and Intelligence."

In the 1950s, the terms "machine intelligence," "artificial intelligence," and "machine learning" all referred to the *goal* of getting "machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence" (to again quote Arthur Samuel).

However, in the intervening five decades, the terms "artificial intelligence" and "machine learning" progressively diverged from their original goal-oriented meaning. These terms are now primarily associated with particular *methodologies* for attempting to achieve the goal of getting computers to automatically solve problems. Thus, the term "artificial intelligence" is today primarily associated with attempts to get computers to solve problems using methods that rely on knowledge, logic, and various analytical and mathematical methods. The term "machine learning" is today primarily associated with attempts to get computers to solve problems that use a particular small and somewhat arbitrarily chosen set of methodologies (many of which are statistical in nature). The narrowing of these terms is in marked contrast to the broad field envisioned by Samuel at the time when he coined the term "machine learning" in the 1950s, the charter of the original founders of the field of artificial intelligence, and the broad vision encompassed by Turing's term "machine intelligence."

Of course, the shift in focus from broad goals to narrow methodologies is an all-too-common sociological phenomenon in academic research.

Turing's term "machine intelligence" did not undergo this arteriosclerosis because, by accident of history, it was never appropriated or monopolized by any group of academic researchers whose primary dedication is to a particular methodological approach. Thus, Turing's term remains catholic today. We prefer to use Turing's term because it still communicates the broad *goal* of getting computers to automatically solve problems in a human-like way.

In his 1948 paper, Turing identified three broad approaches by which human-competitive machine intelligence might be achieved.

The first approach was a logic-driven search. Turing's interest in this approach is not surprising in light of Turing's own pioneering work in the 1930s on the logical foundations of computing.

The second approach for achieving machine intelligence was what he called a "cultural search" in which previously acquired knowledge is accumulated, stored in libraries, and brought to bear in solving a problem—the approach taken by modern knowledge-based expert systems.

Turing's first two approaches have been pursued over the past 50 years by the vast majority of researchers using the methodologies that are today primarily associated with the term "artificial intelligence."

However, most pertinently for this book, Turing also identified a third approach to machine intelligence in his 1948 paper entitled "Intelligent Machinery" (Turing 1948, page 12; Ince 1992, page 127; Meltzer and Michie 1969, page 23), saying:

> "There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being the survival value."

Turing did not specify in 1948 how to conduct the "genetical or evolutionary search" for solutions to problems and, in particular, did not mention the concept of a population or recombination. However, he did point out in his 1950 paper "Computing Machinery and Intelligence" (Turing 1950, page 456; Ince 1992, page 156):

> "We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution, by the identifications
>
> "Structure of the child machine = Hereditary material
>
> "Changes of the child machine = Mutations
>
> "Natural selection = Judgment of the experimenter"

Thus, Turing correctly perceived in 1948 and 1950 that machine intelligence might be achieved by an evolutionary process in which a description of a computer program (the hereditary material) undergoes progressive modification (mutation) under the guidance of natural selection (i.e., selective pressure in the form of what is now usually called "fitness" by practitioners of genetic and evolutionary computation).

Of course, the measurement of fitness in modern genetic and evolutionary computation is usually performed by automated means (as opposed to a human passing judgment on each candidate individual, as suggested by Turing). In addition, modern work generally employs a population (i.e., not just a point-to-point evolutionary progression) and sexual recombination—two key aspects of John Holland's seminal work on genetic algorithms, *Adaptation in Natural and Artificial Systems* (Holland 1975).

1.1.5 Human-Competitiveness of Results Produced by Genetic Programming

The previous four sub-sections defined the terms "human-competitive," "high-return," "routine," and "machine intelligence." In this sub-section (and the next three sub-sections), we evaluate the results produced by genetic programming in light of these four definitions.

Starting with human-competitiveness, table 1.3 lists the 36 human-competitive instances (of which we are aware) where genetic programming has produced

Table 1.3 Thirty-six human-competitive results produced by genetic programming

	Claimed instance	Basis for claim of human-competitiveness	Reference
1	Creation of a better-than-classical quantum algorithm for the Deutsch-Jozsa "early promise" problem	B, F	Spector, Barnum, and Bernstein 1998
2	Creation of a better-than-classical quantum algorithm for Grover's database search problem	B, F	Spector, Barnum, and Bernstein 1999
3	Creation of a quantum algorithm for the depth-two AND/OR query problem that is better than any previously published result	D	Spector, Barnum, Bernstein, and Swamy 1999; Barnum, Bernstein, and Spector 2000
4	Creation of a quantum algorithm for the depth-one OR query problem that is better than any previously published result	D	Barnum, Bernstein, and Spector 2000
5	Creation of a protocol for communicating information through a quantum gate that was previously thought not to permit such communication	D	Spector and Bernstein 2003
6	Creation of a novel variant of quantum dense coding	D	Spector and Bernstein 2003
7	Creation of a soccer-playing program that won its first two games in the Robo Cup 1997 competition	H	Luke 1998
8	Creation of a soccer-playing program that ranked in the middle of the field of 34 human-written programs in the Robo Cup 1998 competition	H	Andre and Teller 1999
9	Creation of four different algorithms for the transmembrane segment identification problem for proteins	B, E	Sections 18.8 and 18.10 of *Genetic Programming II* and sections 16.5 and 17.2 of *Genetic Programming III*
10	Creation of a sorting network for seven items using only 16 steps	A, D	Sections 21.4.4, 23.6, and 57.8.1 of *Genetic Programming III*
11	Rediscovery of the Campbell ladder topology for lowpass and highpass filters	A, F	Section 25.15.1 of *Genetic Programming III* and section 5.2 of this book
12	Rediscovery of the Zobel "*M*-derived half section" and "constant *K*" filter sections	A, F	Section 25.15.2 of *Genetic Programming III*
13	Rediscovery of the Cauer (elliptic) topology for filters	A, F	Section 27.3.7 of *Genetic Programming III*
14	Automatic decomposition of the problem of synthesizing a crossover filter	A, F	Section 32.3 of *Genetic Programming III*
15	Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits	A, F	Section 42.3 of *Genetic Programming III*
16	Synthesis of 60 and 96 decibel amplifiers	A, F	Section 45.3 of *Genetic Programming III*

(Continued)

Table 1.3 *(Continued)*

	Claimed instance	Basis for claim of human-competitiveness	Reference
17	Synthesis of analog computational circuits for squaring, cubing, square root, cube root, logarithm, and Gaussian functions	A, D, G	Section 47.5.3 of *Genetic Programming III*
18	Synthesis of a real-time analog circuit for time-optimal control of a robot	G	Section 48.3 of *Genetic Programming III*
19	Synthesis of an electronic thermometer	A, G	Section 49.3 of *Genetic Programming III*
20	Synthesis of a voltage reference circuit	A, G	Section 50.3 of *Genetic Programming III*
21	Creation of a cellular automata rule for the majority classification problem that is better than the Gacs-Kurdyumov-Levin (GKL) rule and all other known rules written by humans	D, E	Andre, Bennett, and Koza 1996 and section 58.4 of *Genetic Programming III*
22	Creation of motifs that detect the D–E–A–D box family of proteins and the manganese superoxide dismutase family	C	Section 59.8 of *Genetic Programming III*
23	Synthesis of topology for a PID-D2 (proportional, integrative, derivative, and second derivative) controller	A, F	Section 3.7 of this book
24	Synthesis of an analog circuit equivalent to Philbrick circuit	A, F	Section 4.3 of this book
25	Synthesis of a `NAND` circuit	A, F	Section 4.4 of this book
26	Simultaneous synthesis of topology, sizing, placement, and routing of analog electrical circuits	A, F, G	Chapter 5 of this book
27	Synthesis of topology for a PID (proportional, integrative, and derivative) controller	A, F	Section 9.2 of this book
28	Rediscovery of negative feedback	A, E, F, G	Chapter 14 of this book
29	Synthesis of a low-voltage balun circuit	A	Section 15.4.1 of this book
30	Synthesis of a mixed analog-digital variable capacitor circuit	A	Section 15.4.2 of this book
31	Synthesis of a high-current load circuit	A	Section 15.4.3 of this book
32	Synthesis of a voltage-current conversion circuit	A	Section 15.4.4 of this book
33	Synthesis of a Cubic function generator	A	Section 15.4.5 of this book
34	Synthesis of a tunable integrated active filter	A	Section 15.4.6 of this book
35	Creation of PID tuning rules that outperform the Ziegler-Nichols and Åström-Hägglund tuning rules	A, B, D, E, F, G	Chapter 12 of this book
36	Creation of three non-PID controllers that outperform a PID controller that uses the Ziegler-Nichols or Åström-Hägglund tuning rules	A, B, D, E, F, G	Chapter 13 of this book

human-competitive results. Each entry in the table is accompanied by the criteria (from table 1.2) that establish the basis for the claim of human-competitiveness.

This book reports in detail on the last 14 of the human-competitive results in table 1.3. The rating of "human-competitive" is justified for each such result (sections 3.7.3, 4.3.3, 4.4.3, 5.2.3, 9.2.3, 12.4, 13.3, 14.3, and 15.6).

Twenty-three of the instances in table 1.3 involve patents (as indicated by an "A" in column 3). Eleven of the automatically created results infringe previously issued patents and 10 duplicate the functionality of previously patented inventions in a non-infringing way. The 29th through 34th entries in table 1.3 relate to patents for analog circuits that were issued after January 1, 2000. The last two entries are patentable new inventions. Tables C.1 and C.2 in appendix C provides additional details on the 23 patent-related results produced by genetic programming.

1.1.6 High-Return of the Results Produced by Genetic Programming

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 36 results in table 1.3 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 36 results. Answering this question requires the discipline of carefully identifying the amount of analysis, intelligence, information, and knowledge that was supplied by the intelligent human user prior to launching a run of genetic programming.

In this book (and our previous books and papers on genetic programming), we make a clear distinction between the problem-specific preparatory steps and the problem-independent executional steps of a run of genetic programming.

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

The *executional steps* are the problem-independent and domain-independent steps that are automatically executed during a run of the problem-solving method. The *executional steps* of genetic programming are defined by the flowchart in figure 2.1 of this book. The results produced by the executional steps provide the "A" component of the AI ratio (i.e., the numerator).

The five major preparatory steps for the basic version of 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 in the population),
(4) certain parameters for controlling the run, and
(5) a termination criterion and method for designating the result of the run.

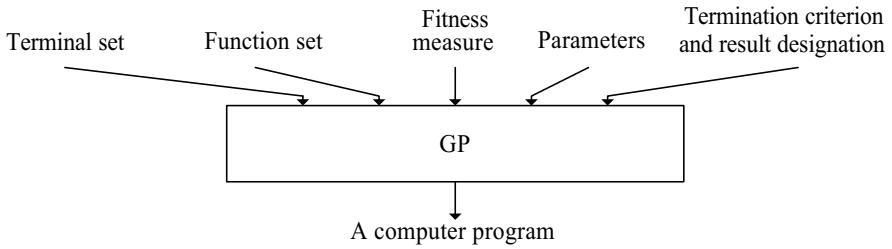

Figure 1.1 Five major preparatory steps for the basic version of genetic programming.

Figure 1.1 shows the five major preparatory steps for the basic version of genetic programming. The preparatory steps (shown at the top of the figure) are the input to the genetic programming system. A computer program (shown at the bottom) is the output of the genetic programming system. The program that is automatically created by genetic programming may solve, or approximately solve, the user's problem.

Genetic programming requires a set of primitive ingredients to get started. The first two preparatory steps specify the primitive ingredients that are to be used to create the to-be-evolved programs. The universe of allowable compositions of these ingredients defines the search space for a run of genetic programming.

The identification of the function set and terminal set for a particular problem (or category of problems) is often a mundane and straightforward process that requires only *de minimus* knowledge and platitudinous information about the problem domain.

For example, if the goal is to get genetic programming to automatically program a robot to mop the entire floor of an obstacle-laden room, the human user must tell genetic programming that the robot is capable of executing functions such as moving, turning, and swishing the mop. The human user must supply this information prior to a run because the genetic programming system does not have any built-in knowledge telling it that the robot can perform these particular functions. Of course, the necessity of specifying a problem's primitive ingredients is not a unique requirement of genetic programming. It would be necessary to impart this same basic information to a neural network learning algorithm, a reinforcement learning algorithm, a decision tree, a classifier system, an automated logic algorithm, or virtually any other automated technique that is likely to be used to solve this problem.

Similarly, if genetic programming is to automatically synthesize an analog electrical circuit, the human user must supply basic information about the ingredients that are appropriate for solving a problem in the domain of analog circuit synthesis. In particular, the human user must inform genetic programming that the components of the to-be-created circuit may include transistors, capacitors, and resistors (as opposed to, say, neon bulbs, relays, and doorbells). Although this information may be second nature to anyone working with electrical circuits, genetic programming does not have any built-in knowledge concerning the fact that transistors, capacitors, and resistors are the workhorse components for nearly all present-day electrical circuits. Once the human user has identified the primitive ingredients, the same function set can be used to automatically synthesize amplifiers, computational circuits, active filters, voltage reference circuits, and any other circuit composed of these basic ingredients.

Likewise, genetic programming does not know that the inputs to a controller include the reference signal and plant output and that controllers are composed of integrators, differentiators, leads, lags, gains, adders, subtractors, and the like. Thus, if genetic programming is to automatically synthesize a controller, the human user must give genetic programming this basic information about the field of control.

The third preparatory step concerns the fitness measure for the problem. The fitness measure specifies what needs to be done. The result that is produced by genetic programming specifies "how to do it." The fitness measure is the primary mechanism for communicating the high-level statement of the problem's requirements to the genetic programming system. If one views the first two preparatory steps as defining the search space for the problem, one can then view the third preparatory step (the fitness measure) as specifying the search's desired direction.

The fitness measure is the means of ascertaining that one candidate individual is better than another. That is, the fitness measure is used to establish a partial order among candidate individuals. The partial order is used during the executional steps of genetic programming to select individuals to participate in the various genetic operations (i.e., crossover, reproduction, mutation, and the architecture-altering operations).

The fitness measure is derived from the high-level statement of the problem. Indeed, for many problems, the fitness measure may be almost identical to the high-level statement of the problem. The fitness measure typically assigns a single numeric value reflecting the extent to which a candidate individual satisfies the problem's high-level requirements. For example:

- If an electrical engineer needs a circuit that amplifies an incoming signal by a factor of 1,000, the fitness measure might assign fitness to a candidate circuit based on how closely the circuit's output comes to a target signal whose amplitude is 1,000 times that of the incoming signal. In comparing two candidate circuits, amplification of 990-to-1 would be considered better than 980-to-1.
- If a control engineer wants to design a controller for the cruise control device in a car, the fitness measure might be based on the time required to bring the car's speed up from 55 to 65 miles per hour. When candidate controllers are compared, a rise time of 10.1 seconds would be considered better than 10.2 seconds.
- If a robot is expected to mop a room, the fitness measure might be based on the percentage of the area of the floor that is cleaned within a reasonable amount of time.
- If a classifier is needed for protein sequences (or any other objects), the fitness measure might be based on the correlation between the category to which the classifier assigns each protein sequence and the correct category.
- If a biochemist wants to find a network of chemical reactions or a metabolic pathway that matches observed data, the fitness measure might assign fitness to a candidate network based on how closely the network's output matches the data.

The fitness measure for a real-world problem is typically multiobjective. That is, there may be more than one element that is considered in ascertaining fitness. For example, the engineer may want an amplifier with 1,000-to-1 gain, but may also want low distortion, low bias, and a low parts count. In practice, the elements of a multiobjective fitness measure usually conflict with one another. Thus, a multiobjective

fitness measure must prioritize the different elements so as to reflect the tradeoffs that the engineer is willing to accept. For example, the engineer may be willing to tolerate an additional 1% of distortion in exchange for the elimination of one part from the circuit. One approach is to blend the distinct elements of a fitness measure into a single numerical value (often merely by weighting them and adding them together).

The fourth and fifth preparatory steps are administrative.

The fourth preparatory step entails specifying the control parameters for the run. The major control parameters are the population size and the number of generations to be run. Some analytic methods are available for suggesting optimal population sizes for runs of the genetic algorithm on particular problems. However, the practical reality is that we generally do not use any such analytic method to choose the population size. Instead, we determine the population size such that genetic programming can execute a reasonably large number of generations within the amount of computer time we are willing to devote to the problem. As for other control parameters, we have, broadly speaking, used the same (undoubtedly non-optimal) set of minor control parameters from problem to problem over a period of years. Although particular problems in this book could possibly be solved more efficiently by means of a different choice of control parameters, we believe that our policy of substantial consistency in the choice of control parameters helps the reader eliminate superficial concerns that the demonstrated success of genetic programming depends on shrewd or fortuitous choices of the control parameters. As can be seen in this book (and our previous books), we frequently make only one run (or, at most, only a few runs) of each major new problem.

The fifth preparatory step consists of specifying the termination criterion and the method of designating the result of the run.

We have now identified that which is supplied by the human user of genetic programming. For the problems in this book, we believe that it is generally fair to say that only a *de minimus* amount of "I" is contained in the problem's primitive ingredients (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 administrative 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 administrative parameters of some kind, and does not contain a termination criterion of some kind.

In view of the numerous human-competitive results produced by genetic programming (table 1.3), it can be seen that genetic programming is capable of delivering a large amount of "A." That is, its AI ratio is high.

Throughout this book, there are numerous sections where the AI ratio is qualitatively evaluated for particular problems (sections 3.7.4, 3.8.4, 3.9.4, 3.10.4, 4.2.4, 4.3.5, 4.4.5, 4.5.4, 4.6.4, 4.7.4, 5.2.5, 5.3.4, 6.7, 7.4.2, 8.6.2, 8.7.2, 9.1.4, 9.2.5, 10.2.4, 10.3.4, 10.4.4, 10.5.4, 11.1.4, 11.2.4, 12.6, 13.5, 14.5, and 15.8).

1.1.7 Routineness of the Results Produced by Genetic Programming

This book demonstrates the generality of genetic programming by solving illustrative problems from several fields, including problems involving

- control,
- analog electrical circuits (including six post-2000 patented circuits),
- placement and routing of circuits,
- antennas,
- genetic networks, and
- metabolic pathways.

Our previous publications (and previous publications by others) additionally demonstrate that genetic programming is capable of solving problems in numerous other areas.

The bright line distinction between that which is delivered by genetic programming and that which is supplied by the intelligent human user (in section 1.1.6) additionally helps make it clear that genetic programming is a systematic general problem-solving method.

As will be seen in this book, relatively little effort is required to make the transition to new problems within a particular domain or to new problems from an entirely different domain.

For example, after discussing the first problem of automatically synthesizing both the topology and tuning of a controller in chapter 3, the transition to each subsequent problem of controller synthesis in that chapter mainly involves providing genetic programming with a different specification of what needs to be done—that is, a different fitness measure. Because 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), additional problems of controller synthesis can be handled merely by changing the statement of what needs to be done.

Similarly, after discussing the first problem of automatically synthesizing both the topology and sizing of an analog electrical circuit in chapter 4, the transition to each subsequent problem of circuit design in that chapter mainly involves providing genetic programming with a different specification of what needs to be done.

The routineness of the transition from problem to problem is especially clear in chapter 15 involving six circuits that were patented after January 1, 2000. All six problems were run consecutively over a period of about two months intentionally using the very same computer, the very same software, and the very same settings of the minor control parameters. All six circuits were composed of the workhorse ingredients of present-day electronics (i.e., resistors, capacitors, and transistors). As we move from one problem to the next in chapter 15, the only substantial change is the specification of what needs to be done. This specification is based on each inventor's statement of performance of each patented circuit. As stated in *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Koza 1992a), "Structure arises from fitness."

The transition from one problem domain to another becomes especially clear by comparing the work concerning the automatic synthesis of controllers, analog electrical circuits, antennas, genetic networks, and networks of chemical reactions.

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

In making the transition from problems of automatic synthesis of circuits to problems of automatic synthesis of networks of chemical reactions (metabolic pathways), the primitive ingredients change to functions that represent chemical reactions that consume chemical substrates (inputs to chemical reactions) and produce reaction products (outputs), at certain rates, in the presence of certain catalysts (enzymes). The fitness measure compares the quantity of product that is produced by a candidate network to the observed data.

Of course, although the preparatory steps change from one problem to another and from one domain to another, the main executional steps (i.e., the flowchart) of genetic programming remain unchanged.

In numerous places throughout this book, we demonstrate

- the routineness of the transition from one problem to another problem in the same domain (sections 3.8.3, 3.9.3, 3.10.3, 4.3.4. 4.4.4, 4.5.3, 4.6.3, 4.7.3, 5.3.3, 8.7.1, 9.2.4, 10.3.3. 10.4.3. 10.5.3, 11.2.3, 13.4. 14.4. and 15.7),
- the routineness of the transition from one domain to the next (sections 4.2.3, 5.2.4, 6.6, 7.4.1, 8.6.1, and 12.5),
- the routineness of the transition from a non-parameterized version of a problem to a parameterized version (sections 9.1.3 and 10.2.3), and
- the routineness of the transition from a problem involving parameterized topologies without conditional developmental operators to a problem involving parameterized topologies with them (section 11.1.3).

1.1.8 Machine Intelligence

As will be seen throughout this book, genetic programming does indeed succeed in getting computers to automatically solve problems from a high-level statement of what needs to be done.

1.2 Genetic Programming Is an Automated Invention Machine

In a commencement address at Worcester Polytechnic Institute in 2000, C. Michael Armstrong, CEO of American Telephone & Telegraph, recounted:

> "On a sweltering summer morning in August 1927, a young man was seated on a passenger ferry as it churned across Upper New York Bay toward Manhattan.

He was gazing idly at the Statue of Liberty when suddenly he jumped from his seat and began frantically searching his pockets for a scrap of paper.

"Coming up empty, he raced to the newsboy on deck and bought a copy of *The New York Times*. The man tore through the pages until he found one that was nearly free of type. He uncapped his fountain pen, sketched a couple of crude diagrams, and surrounded them with mathematical equations."

Holding up the now-famous page from *The New York Times* (figure 1.2), C. Michael Armstrong continued:

"When the ferryboat docked at Manhattan, he raced to his office at Bell Laboratories. He showed his diagrams and equations to one of his coworkers who

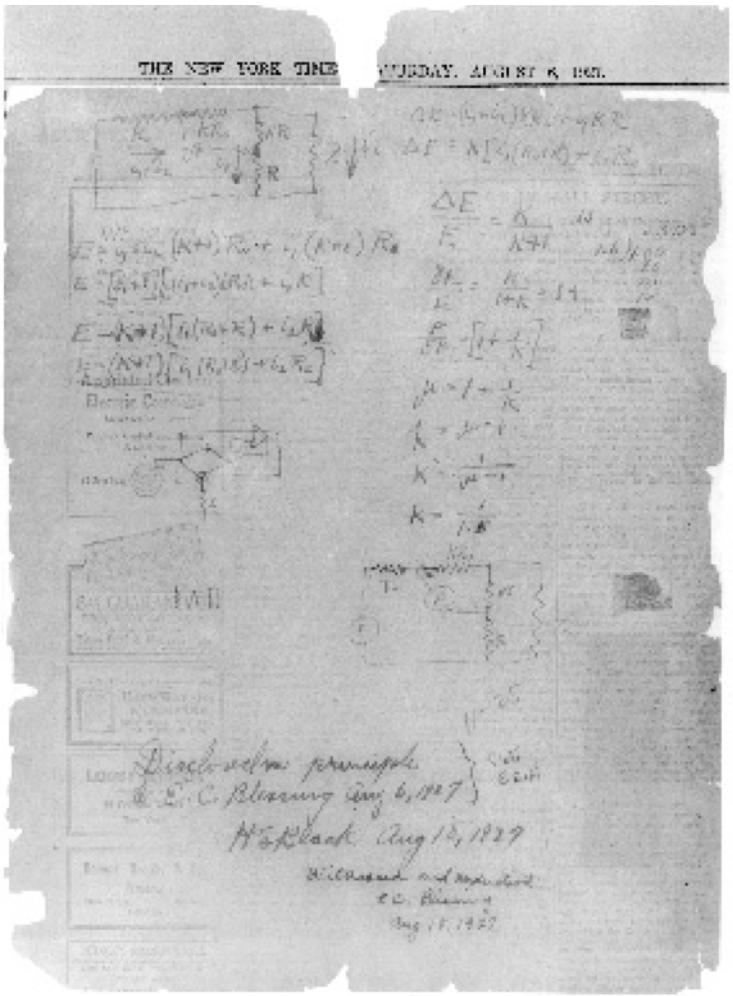

Figure 1.2 Notes written by Harold S. Black on a page of *The New York Times* while commuting on the Lackawanna Ferry. Reproduced here by kind permission of Lucent Technologies.

read them carefully. Then his friend let out a big whoop and they both scrawled their initials on the newspaper page.

"The young man on the ferryboat was Harold Black, Worcester Polytechnic Institute Class of 1921. And the scribblings on his newspaper were the blueprint for the negative-feedback amplifier, a device that played a vital role in 20th century electronics."

Referring to the scribblings on this newspaper page, Mervin Kelly, then president of Bell Labs, said in 1957 (Black 1977):

"Although many of Harold's inventions have made great impact, that of the negative feedback amplifier is indeed the most outstanding. It easily ranks coordinate with De Forest's invention of the audion as one of the two inventions of broadest scope and significance in electronics and communications of the past 50 years … It is no exaggeration to say that without Black's invention, the present long-distance telephone and television networks which cover our entire country and the transoceanic telephone cables would not exist. The application of Black's principle of negative feedback has not been limited to telecommunications. … [T]he entire explosive extension of the area of control, both electrical and mechanical, grew out of an understanding of the feedback principle."

Lee (1998) recounts the history that predated Black's 1927 invention of negative feedback. Earlier work on feedback included rocket pioneer Robert Goddard's 1915 patent for a vacuum tube oscillator using *positive* feedback (Goddard 1915) and Edwin Howard Armstrong's 1914 patent on amplifiers again using positive feedback (Armstrong 1914). As Lee observes,

"[P]rogress in electronics in those early years was largely made possible by Armstrong's regenerative [positive feedback] amplifier, since there was no other economical way to obtain large amounts of gain from the primitive (and expensive) vacuum tubes of the day." …

"Armstrong was able to get gain from a single stage that others could obtain only by cascading several. This achievement allowed the construction of relatively inexpensive, high-gain receivers and therefore also enabled dramatic reductions in transmitter power because of the enhanced sensitivity provided by this increased gain. In short order, *the positive feedback (regenerative) amplifier became a nearly universal idiom*, and Westinghouse (to whom Armstrong had assigned patent rights) kept its legal staff quite busy trying to make sure that only licensees were using this revolutionary technology." (Emphasis added.)

However, Westinghouse's "nearly universal idiom" did not solve a major problem facing American Telephone & Telegraph at the time, namely distortion in amplifiers. As Lee (1998, page 387) further points out:

"Although Armstrong's regenerative amplifier pretty much solved the problem of obtaining large amounts of gain from vacuum tube amplifiers, a different problem

preoccupied the telephone industry. In trying to extend communications distances, amplifiers were needed to compensate for transmission-line attenuation. Using amplifiers available in those early days, distances of a few hundred miles were routinely achievable and, with great care, perhaps 1,000–2,000 miles was possible, but the quality was poor. After a tremendous amount of work, a crude transcontinental telephone service was inaugurated in 1915, with a 68-year-old Alexander Graham Bell making the first call to his former assistant, Thomas Watson, but this feat was more of a stunt than a practical achievement.

"The problem wasn't one of insufficient amplification; it was trivial to make the signal at the end of the line quite loud. Rather the problem was distortion. Each amplifier contributed some small (say, 1%) distortion. Cascading a hundred of these things guaranteed that what came out didn't very much resemble what went in.

"The main 'solution' at the time was to (try to) guarantee 'small signal' operation of the amplifiers. That is, by restricting the dynamic range of the signals to a tiny fraction of the amplifier's overall capability, more linear operation could be achieved. Unfortunately, this strategy is quite inefficient since it requires the construction of, say, 100-W amplifiers to process milliwatt signals. Because of the arbitrary distance between a signal source and an amplifier (or possibly between amplifiers), though, it was difficult to guarantee that the input signals were always sufficiently small to satisfy linearity."

Such was the state of affairs when Harold S. Black started working in 1921 at AT&T on the problem of reducing amplifier distortion. After considerable work, Black reached the conclusion in 1923 (Black 1977):

"There was just no way to meet our ambitious goal."

As Black recounts:

"This might have been the end of it, except that, on March 16, 1923, I was fortunate enough to attend a lecture by the famous scientist and engineer, Charles Proteus Steinmetz." …

"I no longer remember the subject, but I do remember the clarity and logic of his presentation." …

"I was so impressed by how Steinmetz got down to the fundamentals that when I returned home at 2 A.M., I restated my own problems as follows: Remove all distortion products from the amplifier output. In doing this, I was accepting an imperfect amplifier and regarding its output as composed of what was wanted plus what was not wanted. I considered what was not wanted to be distortion (regardless of whether it was due to nonlinearity, variation in the tube gain, or whatever), and I asked myself how to isolate and then eliminate this distortion. I immediately observed that by reducing the output to the same amplitude as the input, and subtracting one from the other, only the distortion would remain. This distortion could then be amplified in a separate amplifier and used to cancel out the distortion in the original amplifier. …

"The next day, March 17, I sketched two such embodiments and thereby invented the feed-forward amplifier. …

"Later that day, I set up each embodiment in the laboratory. Both worked as expected."

Black applied for a patent on his 1923 invention of the feed-forward amplifier and the patent was issued in 1928 (Black 1928). Unfortunately, his 1923 invention did not turn out to be practical. As Black (1977) laments,

> "[T]he invention required precise balances and subtractions that were hard to achieve and maintain with the amplifiers available at that time. …
>
> "Over the next four years, I struggled with the problem of turning my intention into an amplifier that was practical. …
>
> "[F]or my purpose the gain had to be absolutely perfect."
>
> "For example, every hour on the hour—24 hours a day—somebody had to adjust the filament current to its correct value. …
>
> "In addition, every six hours it became necessary to adjust the B battery voltage, because the amplifier gain would get out of hand.
>
> "There were other complications too, but these were enough!"

The bottom line concerning the feed-forward amplifier that Black invented in 1923 was:

> "Nothing came of my efforts, however, because every circuit I devised turned out to be far too complex to be practical."

In spite of this false start, Black continued to work on the problem.

After working on the problem for a total of six years:

> "Then came the morning of Tuesday, August 2, 1927, when the concept of the negative feedback amplifier came to me in a flash while I was crossing the Hudson River on the Lackawanna Ferry, on my way to work. For more than 50 years, I have pondered how and why the idea came, and I can't say any more today than I could that morning. All I know is that after several years of hard work on the problem, I suddenly realized that if I fed the amplifier output back to the input, in reverse phase, and kept the device from oscillating (singing, as we called it then), I would have exactly what I wanted: a means of canceling out the distortion of the output. I opened my morning newspaper and on a page of *The New York Times* I sketched a simple canonical diagram of a negative feedback amplifier plus the equations for the amplification with feedback. I signed the sketch, and 20 minutes later, when I reached the laboratory at 463 West Street, it was witnessed, understood, and signed by the late Earl C. Blessing."

Numerous other inventors have reported similar singular moments when their previous thinking about a vexatious problem crystallized into an invention.

1.2.1 The Illogical Nature of Invention and Evolution

Most computer scientists unquestioningly assume that any effective problem-solving process must be logically sound and deterministic.

The consequence of this unproven assumption is that virtually all conventional approaches to artificial intelligence and machine learning possess these characteristics. Yet the reality is that logic does not govern two of the most important processes

for solving complex problems—namely the invention process (performed by creative humans) and the evolutionary process (occurring in nature).

Moreover, neither the invention process nor the evolutionary process is deterministic.

A new idea that can be logically deduced from facts that are known in a field, using transformations that are known in a field, is not considered to be inventive by the Patent Office. A new idea is patentable only if there is what the courts have called an "illogical step" (i.e., a logically unjustified step). The required illogic distinguishes the proposed invention from that which is readily deducible from what is already known. The required illogical step is also sometimes referred to as a "flash of genius." In other words, logical thinking is not the key ingredient for one of the most significant human problem-solving activities, namely the invention process. Interestingly, everyday usage parallels the law concerning the point that a lack of logic is a precondition for inventiveness: People who mechanically apply existing facts in well-known ways are summarily dismissed as being uncreative.

Of course, when we say that the invention process is inherently illogical, we do not mean that logical thinking is not helpful to inventors or that inventors are oblivious to logic. Logical thinking often plays the important role of setting the stage for an invention. "[S]everal years of hard work on the problem" brought Black's thinking into the proximity of a solution (Black 1977). Then, at the critical moment, Black made the illogical leap during his now-famous ferryboat ride. Although logical thinking may play a role in invention and creativity, at the end of the day, the critical element is a logical discontinuity from established ideas.

The design of complex entities by the evolutionary process in nature is another important type of problem-solving that is not governed by logic. In nature, solutions to design problems are discovered by means of evolution and natural selection. The evolutionary process is probabilistic, rather than deterministic. Moreover, it is certainly not guided by mathematical logic. Indeed, one of the most important characteristics of the evolutionary process is that it intentionally creates and actively maintains inconsistent and contradictory alternatives. Logically sound systems do not do that. The active maintenance of inconsistent and contradictory alternatives (called *genetic diversity*) is a precondition for the success of the evolutionary process.

1.2.2 Overcoming Established Beliefs

As previously mentioned, Edwin Howard Armstrong's approach to amplification using positive feedback was "a nearly universal idiom" during the early part of the 20th century.

In spite of the elegance and manifest effectiveness of negative feedback, Armstrong's approach was so entrenched in the thinking of electrical engineers that there was widespread resistance to Black's concept of negative feedback for many years after its invention. As Black (1977) recalls:

> "Although the invention had been submitted to the U.S. Patent Office on August 8, 1928, more than nine years would elapse before the patent was issued on December 21, 1937.... One reason for the delay was that *the concept was so contrary to established beliefs*." (Emphasis added.)

The British Patent Office was even more resistant. As Black (1977) recounted:

> "…our patent application was treated in the same manner as one for a perpetual motion machine."

The British Patent Office continued to maintain that negative feedback would not work in spite of the fact that AT&T had "70 amplifiers working successfully in the telephone building at Morristown" for a number of years.

We believe that one reason why it took an inordinate amount of time for negative feedback to gain acceptance was that human thinking often becomes channeled along the well-traveled paths of "established beliefs."

One of the virtues of genetic programming is that it is not aware, much less concerned, about whether a solution is "contrary to established beliefs." Genetic programming approaches a problem in an open-ended way that is not encumbered by previous human thinking. For this reason, genetic programming often unearths solutions that might have never occurred to human scientists and engineers who are steeped in the thinking of the day.

In the section entitled "Genetic Programming Takes a Ride on the Lackawanna Ferry" (section 14.1), genetic programming is used to reinvent negative feedback. As will be seen, if one begins with a high-level statement of the problem that Black was trying to solve, Black's solution flows almost effortlessly from a run of genetic programming. It does so because Black's solution is a correct solution to the problem and, as they say, necessity is the mother of invention.

For the 20 other instances in table 1.3 where genetic programming created an entity that infringes a previously issued patent or duplicates the functionality of a previously patented invention in a non-infringing or novel way, the solution similarly flowed directly from a high-level statement of the problem.

The 23 instances where genetic programming has duplicated the functionality of a previously patented invention, infringed a previously issued patent, or created a patentable new invention are shown in tables C.1 and C.2 in appendix C.

1.2.3 Automating the Invention Process

For over 200 years, the U.S. Patent Office has been in the business of receiving written descriptions of human-designed inventions and judging whether the purported inventions are

- "new,"
- "improved,"
- "useful," and
- "[un]obvious…to a person having ordinary skill in the art to which said subject matter pertains." (35 *United States Code* 103a)

When the Patent Office passes judgment on a patent application, it generally works from written documents and operates at arms length from the inventor. When an automated method duplicates the detailed structure of a previously patented human-created invention, the fact that the human-designed version originally satisfied

the Patent Office's criteria for patent-worthiness means that an automatically created duplicate would also have satisfied the Patent Office's criteria for patent-worthiness had it arrived at the Patent Office prior to the human inventor's submission.

When genetic programming is applied to a problem whose solution is a previously patented invention, there are three possible outcomes:

- failure of the run to solve the problem,
- creation of a solution that infringes a previously issued patent, or
- creation of a non-infringing solution that duplicates the functionality of a previously patented invention.

There are two sub-cases associated with the third case.

First, a non-infringing solution may be a previously known solution (i.e., prior art). The previously known solution may or may not have been patented in the past.

Second, a non-infringing solution may be a new solution to the problem.

In this second sub-case, a new, genetically evolved, non-infringing solution may be patentable if it satisfies the additional requirements of being "useful," "improved," and "unobvious."

A genetically evolved solution would generally be deemed to be "useful" for the same reasons that the originally patented invention was deemed to be "useful."

Almost every alternative solution to a particular problem usually has some attribute that can be reasonably viewed (from some standpoint) as being "improved" in some respect or to some degree.

Because genetically evolved solutions often contain features that would never occur to human scientists or engineers, a genetically evolved alternative solution will often be "unobvious" to someone "having ordinary skill in the art."

U.S. law suggests that inventions created by automated means are patentable by saying:

> "Patentability shall not be negatived by the manner in which the invention was made." (35 *United States Code* 103a)

1.2.4 Patentable New Inventions Produced by Genetic Programming

Given that genetic programming has solved problems whose solutions were previously patented, it is a natural extension to try to use genetic programming to generate patentable new inventions.

Chapters 12 and 13 of this book describe a patent application filed on July 12, 2002, for improved PID (proportional, integrative, and derivative) tuning rules and non-PID controllers that were automatically created by means of genetic programming (Keane, Koza, and Streeter 2002a). The genetically evolved tuning rules and controllers outperform controllers tuned using the widely used Ziegler-Nichols tuning rules (1942) and the recently developed Åström-Hägglund tuning rules (1995). The applicants believe that the new tuning rules and controllers satisfy the statutory requirement of being "improved" and "useful." They are certainly "new." Because they contain features that would never occur to an experienced control engineer, they are certainly "unobvious" to someone "having ordinary skill in the art."

If (as expected) a patent is granted, it will (we believe) be the first patent granted for an invention created by genetic programming. For further discussion of the potential of genetic programming as an invention machine, see Koza, Keane, and Streeter 2003.

1.3 Genetic Programming Can Automatically Create Parameterized Topologies

Eleven problems in this book illustrate this book's third main point, namely that genetic programming can automatically create what we call *parameterized topologies*. That is, genetic programming can automatically create, in a single run, a general (parameterized) solution to a problem in the form of a graphical structure whose nodes or edges represent components and where the parameter values of the components are specified by mathematical expressions containing free variables.

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

As an example, suppose the goal is to design a circuit to feed the woofer speaker of a hi-fi system. That is, the desired circuit is intended to pass signals below a certain frequency at full power into the woofer, but to suppress all higher frequencies. Moreover, suppose that you want a general solution to this design problem. That is, suppose you want a solution that works for any cutoff frequency *f*—not just a solution that works for, say, 1,000 Hertz. A genetically evolved general solution to this problem is shown later in this book (in figure 10.9). The general solution produced by genetic programming includes the circuit's topology. The genetically evolved parameterized circuit has nine components. The general solution includes the type of each of the nine components. There are five capacitors and four inductors in figure 10.9. The connections between the nine components are automatically created during the run of genetic programming. The problem's free variable, *f*, is an input to the genetically evolved solution. The genetically evolved solution is general because the capacitance of the five capacitors and the inductance of the four inductors are not constant, but instead, are functions of the free variable *f*. That is, the general solution

produced by genetic programming includes nine different mathematical expressions—each containing the free variable *f*. For example, one of the nine mathematical expressions is

$$C2 = \frac{1.6786 \times 10^5}{f}$$

When all nine mathematical expressions are instantiated with a particular value of the free variable, *f*, the resulting circuit is a lowpass filter whose passband boundary is *f*. The numerical values of certain components could be constant (although none of the values happen to be constant in this particular case). The advantage of a parameterized topology is it is a general solution to the problem—not just a solution to a single instance of the problem.

If the genetically evolved program additionally contains conditional developmental operators, different graphical structures will, in general, be produced for different instantiations of the free variable. That is, the genetically evolved program will operate as a genetic switch. Depending on the values of the free variable, different graphical structures will result from the execution of the best-of-run program. The numerical values for all parameterized components in the graphical structure will also be established by the execution of the program.

The capability of genetic programming to automatically create parameterized topologies is demonstrated in this book by the automatic synthesis of

- a parameterized controller for controlling a three-lag plant whose time constant is specified by a free variable (section 9.1),
- a parameterized controller for controlling plants belonging to two different families (section 9.2),
- three parameterized controllers for controlling industrially representative plants (chapter 13),
- a parameterized circuit-constructing program tree containing two free variables that yields a Zobel network (section 10.2),
- a parameterized circuit-constructing program tree that yields a passive third-order elliptic lowpass filter whose modular angle is specified by a free variable (section 10.3),
- a parameterized circuit-constructing program tree that yields an active lowpass filter whose passband boundary is specified by a free variable (section 10.5),
- a parameterized circuit-constructing program tree that yields a passive lowpass filter whose passband boundary is specified by a free variable (section 10.4),
- a parameterized circuit-constructing program tree containing conditional developmental operators and free variables that yields either a lowpass or highpass passive filter (section 11.1),
- a parameterized circuit-constructing program tree containing conditional developmental operators and free variables that yields either a lowpass passive filter with a variable passband boundary or a highpass passive filter with a variable passband boundary (section 11.2),
- a parameterized circuit-constructing program tree containing conditional developmental operators and free variables that yields either a quadratic or cubic computational circuit (section 11.3), and

- a parameterized circuit-constructing program tree containing conditional developmental operators and free variables that yields either a 40 dB or 60 dB amplifier (section 11.4).

These 11 examples establish this book's third main point.

1.4 Historical Progression of Qualitatively More Substantial Results Produced by Genetic Programming in Synchrony with Increasing Computer Power

Numerous questions naturally arise in connection with any proposed approach to machine intelligence.

- 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)?
  - Was the method 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?
- 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 discussed in this book establish that it is not toyware.

This book's fourth main point is based on a historical perspective on the progression of results produced by genetic programming over the 15-year period between 1987 and 2002.

Table 1.4 (described in greater detail in section 18.1) lists the five computer systems used to produce our group's reported work on genetic programming in the 15-year period between 1987 and 2002. Column 7 shows the number of human-competitive results (as defined in table 1.2 and itemized in table 1.3) generated by each computer system.

The first entry in the table 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. That is, genetic programming is not serialware.

Table 1.4 shows the following:

- There is an order-of-magnitude increase speed-up (column 4) between each successive computer system in the table. Note that, according to Moore's law (Moore 1996), 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 used for most of the work in this book) 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.

Four successive order-of-magnitude increases in computer power are explicitly shown in table 1.4. An additional order-of-magnitude increase was achieved by the expedient of making extraordinarily long runs on the largest machine in the table (the 1,000-node Pentium® II parallel machine). The length of the run that produced

Table 1.4 Number of human-competitive results produced by genetic programming with five computer systems between 1987 and 2002

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

the genetically evolved controller described in section 13.2.3 was 28.8 days—almost an order-of-magnitude increase (9.3 times) over the 3.4-day average for other problems described in this book (table 17.1). A patent application was filed for the controller produced by this four-week run (Keane, Koza, and Streeter 2002a). This genetically evolved controller outperforms controllers employing the widely used Ziegler-Nichols tuning rules and the recently developed Åström-Hägglund tuning rules. If the final 9.3-to-1 increase in table 1.5 is counted as an additional speed-up, the overall speed-up between the first and last entries in the table is 130,660-to-1.

Table 1.5 (described in greater detail in sections 18.2 and 18.3) is organized around the five just-explained order-of-magnitude increases in the expenditure of computing power. Column 4 of table 1.5 characterizes the qualitative nature of the

Table 1.5 Progression of qualitatively more substantial results produced by genetic programming in relation to five order-of-magnitude increases in computational power

System	Period of usage	Speed-up over previous row in this table	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

results produced by genetic programming. The table shows the progression of qualitatively more substantial results produced by genetic programming in terms of five order-of-magnitude increases in the expenditure of computational resources.

The order-of-magnitude increases in computer power shown in table 1.5 correspond closely (albeit not perfectly) with the following progression of qualitatively more substantial results produced by genetic programming:

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

In other words, genetic programming is able to take advantage of the exponentially increasing computational power made available by iterations of Moore's law—that is, it is Mooreware.

These results (explained in greater detail in chapter 18) establish this book's fourth main point: 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.