EVOLUTIONARY
COMPUTATION
IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Mohamed E. El-Hawary, Editor in Chief

M. Akay       T. G. Croda       M. S. Newman
J. B. Anderson R. J. Herrick     F. M. B. Pereira
R. J. Baker    S. V. Kartalopoulos C. Singh
J. E. Brewer   M. Montrose      G. Zobrist

Kenneth Moore, Director of IEEE Book and Information Services (BIS)
Catherine Faduska, Acquisitions Editor, IEEE Press
Jeanne Audino, Project Editor, IEEE Press

IEEE Computational Intelligence Society, Sponsor
IEEE CI-S Liaison to the IEEE Press, David B. Fogel

Books in the IEEE Press Series on Computational Intelligence

Computationally Intelligent Hybrid Systems
Edited by Seppo J. Ovaska
2005  0-471-47668-4

Handbook of Learning and Approximate Dynamic Programming
Edited by Jennie Si, Andrew G. Barto, Warren B. Powell, Donald Wunsch II
2004  0-471-66054-X

Computational Intelligence: The Experts Speak
Edited by David B. Fogel and Charles J. Robinson
2003  0-471-27454-2
EVOLUTIONARY COMPUTATION
Toward a New Philosophy of Machine Intelligence

Third Edition

DAVID B. FOGEL

IEEE Computational Intelligence Society, Sponsor

IEEE Press Series on Computational Intelligence
David B. Fogel, Series Editor

IEEE PRESS

A JOHN WILEY & SONS, INC., PUBLICATION
Preface to the Third Edition ix
Preface to the Second Edition xi
Preface to the First Edition xv

1 Defining Artificial Intelligence 1

1.1 Background 1
1.2 The Turing Test 3
1.3 Simulation of Human Expertise 5
  1.3.1 Samuel’s Checker Program 6
  1.3.2 Chess Programs 8
  1.3.3 Expert Systems 11
  1.3.4 A Criticism of the Expert Systems or Knowledge-Based Approach 13
  1.3.5 Fuzzy Systems 15
  1.3.6 Perspective on Methods Employing Specific Heuristics 16
1.4 Neural Networks 17
1.5 Definition of Intelligence 21
1.6 Intelligence, the Scientific Method, and Evolution 23
1.7 Evolving Artificial Intelligence 26
   References 27
   Chapter 1 Exercises 31

2 Natural Evolution 33

2.1 The Neo-Darwinian Paradigm 33
2.2 The Genotype and the Phenotype: The Optimization of Behavior 34
2.3 Implications of Wright’s Adaptive Topography: Optimization Is Extensive Yet Incomplete 38
2.4 The Evolution of Complexity: Minimizing Surprise 40
2.5 Sexual Reproduction 41
2.6 Sexual Selection 43
2.7 Assessing the Beneficiary of Evolutionary Optimization 44
Ten years have elapsed since the first publication of this book. In that decade of time, evolutionary computation has matured from a fringe element of computer science to a well-recognized serious endeavor. Although specific numbers are difficult to estimate, it would not be unreasonable to believe that over 10,000 papers have now been published in evolutionary computation. In 2001, the *IEEE Transactions on Evolutionary Computation* placed fourth in impact factor ratings in the category of computer science/artificial intelligence among all such journals that are analyzed by the Institute for Scientific Information (Thomson ISI), and it continues to place among the top publications year after year, while the number of submissions to the journal continues to increase. It has taken some time to achieve this degree of recognition, and it is certainly welcome.

Five years ago, I published the second edition of this book, and noted that evolutionary computation and artificial intelligence (AI) remained mostly disparate activities. Unfortunately, that is still true. Furthermore, other alternative approaches to machine intelligence, such as neural networks and fuzzy systems, still remain outside the mainstream of AI. Surely, these areas are well known, but not as well practiced within what constitutes “traditional” AI.

In contrast, within engineering organizations such methods are being embraced. This is particularly true within the Institute of Electrical and Electronics Engineers (IEEE), the world’s largest professional organization. The IEEE recently approved a request by the IEEE Neural Networks Society to change its name to the IEEE Computational Intelligence Society, emphasizing the important contributions from fuzzy logic, evolutionary computation, and other branches of machine intelligence based on inspiration from nature. At the time of this writing, there are over 5,000 members in this new computational intelligence society. Publications often illustrate the practical application of computational intelligence tools to challenging real-world problems, and there could be no greater evidence of success.

Much of what I offered in the preface to the second edition still holds for this third edition. I encourage you to read that preface, as well as the preface to the first edition, to review the motivation that underscores the philosophy offered in this
book. This new edition has several revisions and additions that highlight recent research and references, as well as an expanded Chapter 5, which provides the most up-to-date review of research on using evolutionary computation to allow a computer to teach itself to play checkers and chess. I thank the IEEE for allowing me, under its copyright provisions, to reprint sections of Chellapilla and Fogel (1999a, 1999b, 2001) and Fogel et al. (2004) in that chapter. In addition, each chapter is now followed by a series of questions and sometimes also programming activities aimed at the student or practitioner. Some of the questions are offered for consideration, and there may be no perfect answer; others involve mathematical development or experimental research. I encourage you to give all these questions and activities your attention.

I would like to thank Cathy Faduska and Chrissy Kuhnen at IEEE for their support during this project, as well as the technical reviewers, and especially my brother, Gary, for his assistance in revising the material in Chapter 2. As with the earlier editions, the purpose of the book will be served if it continues to advance interest, experimentation, and analysis in the growing field of evolutionary computation. Despite the explosion of activity in evolutionary computation, there is still much to learn and much to do, and I hope this book helps you advance the frontier of what evolutionary computation is capable of achieving.

REFERENCES


David B. Fogel

La Jolla, California
September 2005
Evolutionary computation is enjoying a renaissance. Conservatively, well over 2000 papers have been published in evolutionary computation since 1995 when the first edition of this book appeared and became a featured selection of the Library of Science Book Club. Publications on simulated evolution are now part of mainstream science and are no longer something out of the ordinary. Several evolutionary computation journals are now available, with some reaching thousands of readers worldwide (for example, the *IEEE Transactions on Evolutionary Computation*). The prospect of pursuing research in evolutionary computation has never been better.

Despite this visibility and acceptance, evolutionary computation and artificial intelligence (AI) still remain mostly disparate endeavors. The traditional perspective of programming human knowledge into a computer fact by fact, inference by inference, endures.

The crowning achievement of this approach was evidenced in May 1997, when the program Deep Blue defeated Garry Kasparov, the world chess champion. It was a triumph not only of human ingenuity, but also of the speed of computers for searching many moves ahead in the game, the design of application-specific hardware, and hand-tuned evaluation functions to accomplish by brute force what humans do with elegance. It was not, however, a triumph of AI. Indeed, the program that defeated Kasparov was not intelligent in any real sense of the word. The difference between Deep Blue and a calculator is only superficial: Beneath the facade is a preprogrammed “canned” procedure that learns nothing about the world in which it interacts. The headlines that read “Computer Beats Champion” were misleading. A more accurate account would have been “Humans Devise Tool to Beat Champion”—the tool simply happened to be a computer program. With respect to the intelligence of that program, it could just as well have been a hammer.

Deep Blue, like the majority of efforts in AI, simulates symptoms of intelligent behavior as observed in humans. The benefit of this approach is that it can generate highly optimized behavior, even to the extent that it surpasses human perfor-
performance. The drawbacks are two-fold. The most obvious shortcoming is that this optimized behavior is applicable in only a very limited domain. The more fundamental flaw is that this approach does not contribute to an understanding of the intelligence that we observe in humans, animals, social groups, or even in the evolution of life itself.

Intelligence can be defined in terms of the capability of a system to adapt its behavior to meet its goals in a range of environments. The immediate question is then: How intelligent is Deep Blue? It plays an outstanding game of chess but can it play checkers? Can it play tic-tac-toe? Can it discover new strategies for negotiating treaties when the parties involved have conflicting goals? The fact that these are rhetorical questions is evidence of how little has really been accomplished.

Genesereth and Nilsson (1987) suggested that the ultimate goal of AI is a theory that both explains the behavior of naturally occurring intelligent entities and indicates the manner in which artificial entities can be designed to yield intelligent behavior. The argument offered in this book is that the process of evolution accounts for such behavior and provides the foundation for the design of artificially intelligent machines.

In support of this thesis, I offer several examples. In each case, beginning with very restricted knowledge of the environment, an evolutionary algorithm learns how to solve the task at hand. This same algorithm can solve problems with a reasonable level of competency even when the conditions change. To make this point more evident, I have removed a previous example concerning the control of an unstable cart-pole system and have substituted new results of evolving strategies for playing checkers. The reader will note the relatively minor modifications that were required to transition from playing tic-tac-toe, as offered in the first edition and recapitulated here, to playing this new game. The versatility of the evolutionary procedure is one of its main strengths in serving as a basis for generating intelligent behavior, adapting to new challenges, and learning from experience.

This second edition also includes a significantly expanded fourth chapter concerning mathematical and empirical properties of evolutionary algorithms. There have been several important contributions to the theory of evolutionary computation since the publication of the first edition. Two of these have been offered by David Wolpert and William Macready. The first established that no evolutionary algorithm can be superior for all possible problems (i.e., the so-called “no free lunch” theorem), and the second identified a flaw in previous theory that served as the foundation for much work in one subset of evolutionary computation. This edition reviews these novel findings, together with other recent discoveries, so that the student of evolutionary computation can have a ready reference for the current state of knowledge in the field.

This book is intended not only for students and practitioners of computer science and engineering, but also for the general reader who is curious about the prospects of creating intelligent machines. Its purpose will be served if it continues to advance interest, experimentation, and analysis in the growing field of evolutionary computation.
REFERENCE


DAVID B. FOGEL

*Natural Selection, Inc.*
Intelligence and evolution are intimately connected. Intelligence is a natural part of life. It is also, however, a mechanical process that can be simulated and emulated. Intelligence is not a property that can be limited philosophically solely to biological structures. It must be equally applicable to machines. Although such efforts for generating intelligence in machines are typically described by the term artificial intelligence, this is in fact a misnomer. If intelligence were to be properly represented as to its process, there would be nothing artificial about it. If the process is understood, methods for its generation should converge functionally and become fundamentally identical, relying on the same physics whether the intelligence occurs in a living system or a machine.

The majority of research in artificial intelligence has simulated symptoms of intelligent behavior as observed in humans. In contrast, I propose a definition of intelligence that does not rely only in its comparisons to human behavior. Intelligence is defined as the capability of a system to adapt its behavior to meet its goals in a range of environments. The form of the intelligent system is irrelevant, for its functionality is the same whether intelligence occurs within an evolving species, an individual, or a social group. Rather than focus on ourselves and attempt to emulate our own behaviors and cognitive processes, it would appear more prudent to recognize that we are products of evolution and that by modeling evolutionary processes we can create entities capable of generating intelligent behavior. Evolutionary computation, the field of simulating evolution on a computer, provides the basis for moving toward a new philosophy of machine intelligence.

Evolution is categorized by several levels of hierarchy: the gene, the chromosome, the individual, the species, and the ecosystem. Thus there is an inevitable choice that must be made when constructing a simulation of evolution. Inevitably, attention must be focused at a particular level in the hierarchy, and the remainder of the simulation is to some extent determined by that perspective. Ultimately, the question that must be answered is, “What exactly is being evolved?” All the symptoms of the simulation will reflect the answer to this question, whether by conscious design or not. If intelligence is viewed in terms of adaptation, then the answer to the
question must be that what is evolved is functional behavior. To construct a useful model, evolution must be abstracted in terms of the behavioral relationships between units of evolution, rather than the mechanisms that give rise to these relationships.

The result of such modeling is a series of optimization algorithms that rely on very simple rules. These various procedures are implemented as population-based searches over a fitness response surface. The optimization process inherent in selection iteratively improves the quality of these solutions. The rate of improvement is primarily determined by the shape of the response surface, but the procedures generally converge to near-optimal solutions despite the existence of topological pathologies. In some cases, it is provable that the procedures will asymptotically converge to the best solutions; in others, it is provable that they will never converge at all. Nonetheless, these methods offer potential for addressing engineering problems that have resisted solution by classic techniques.

More importantly, however, these methods may be used to address a range of problems rather than any one specific problem. They have proven themselves to be robust and may be applied toward general problem solving. This latter attribute represents the greatest potential for evolutionary computation, yet there have been few investigations in this regard. The real promise of evolutionary computation remains mostly unfulfilled.

This book is an attempt to integrate the inspiration, philosophy, history, mathematics, actualizations, and perspectives of evolutionary computation. It will have been successful if this integration adds to the reader’s understanding of the field of research. My hope is that it will encourage further investigation into the potential of these techniques for generating machine intelligence.

ACKNOWLEDGMENTS

There are many people to thank for their time and effort spent helping me prepare the manuscript and offering comments on this work. Dudley Kay, Lisa Mizrahi, Val Zaborski, and Karen Henley of IEEE Press and copyeditor Kathleen Lafferty were most supportive. Teresa Moniz and Mike MacVittie assisted with the artwork, and I would like to thank them and Orincon Corporation, and especially Terry Rickard, for making their time and skills available. I also greatly appreciate the help received from Scott Haffner, Brian Moon, and Rob Redfield in converting and debugging computer programs for execution on different machines. Chapters and sections, including those taken from recent publications, were reviewed by Lee Altenberg, Russell Anderson, Pete Angeline, Valerie Atmar, Wirt Atmar, Thomas Back, Rick Barton, Hans Bremermann, Tom Brotherton, Cliff Brunk, George Burgin, Michael Conrad, Lawrence (Dave) Davis, Hugo de Garis, Ken De Jong, Marco Dorigo, Tom English, Gary Fogel, Lany Fogel, Steve Frank, John Grefenstette, Paul Harrald, David Honig, Gerry Joyce, Mark Kent, Avi Krieger, Charles Marshall, Ernst Mayr, Zbyszek Michalewicz, Mike O’Hagan, Bill Porto, Bob Powell, Tom Ray, Ingo Rechenberg, Bob Reynolds, Matt Rizki, Stuart Rubin, Günter Rudolph, N. (Sara-
van) Saravanan, Bill Schopf, Hans-Paul Schwefel, Brad Scurlock, Tony Sebald, Pat Simpson, Chris Wills, Xin Yao, and several anonymous referees. Their criticisms have been invaluable. I would be remiss if I did not acknowledge the encouragement of John McDonnell, Ward Page, and Jennifer Schlenzig, and I am deeply grateful to Wirt Atmar and Tony Sebald for years of guidance and friendship. Finally, I thank my family for their constant support.

I would like to thank IEEE Press for encouraging a second edition of this work and, in particular, I am grateful to John Griffin, Marilyn Catis, Ken Moore, Mark Morrell, and Barb Soifer for their support. I would also like to thank the six technical reviewers for this edition, as well as Pete Angeline, Gary Fogel, Larry Fogel, and Bill Porto, for their advice and criticism.

DAVID B. FOGEL

Natural Selection, Inc.
1.1 BACKGROUND

Calculators are not intelligent. Calculators give the right answers to challenging math problems, but everything they “know” is preprogrammed by people. They can never learn anything new, and outside of their limited domain of utility, they have the expertise of a stone. Calculators are able to solve problems entirely because people are already able to solve those same problems.

Since the earliest days of computing, we have envisioned machines that could go beyond our own ability to solve problems—intelligent machines. We have generated many computing devices that can solve mathematical problems of enormous complexity, but mainly these too are merely “calculators.” They are preprogrammed to do exactly what we want them to do. They accept input and generate the correct output. They may do it at blazingly fast speeds, but their underlying mechanisms depend on humans having already worked out how to write the programs that control their behavior. The dream of the intelligent machine is the vision of creating something that does not depend on having people preprogram its problem-solving behavior. Put another way, artificial intelligence should not seek to merely solve problems, but should rather seek to solve the problem of how to solve problems.

Although most scientific disciplines, such as mathematics, physics, chemistry, and biology, are well defined, the field of artificial intelligence (AI) remains enigmatic. This is nothing new. Even 20 years ago, Hofstadter (1985, p. 633) remarked, “The central problem of AI is the question: What is the letter ‘a’? Donald Knuth, on hearing me make this claim once, appended, ‘And what is the letter ‘i’?’—an amendment that I gladly accept.” Despite nearly 50 years of research in the field, there is still no widely accepted definition of artificial intelligence. Even more, a discipline of computational intelligence—including research in neural networks, fuzzy systems, and evolutionary computation—has gained prominence as an alternative to AI, mainly because AI has failed to live up to its promises and because many believe that the methods that have been adopted under the old rubric of AI will never succeed.

It may be astonishing to find that five decades of research in artificial intelligence have been pursued without fundamentally accepted goals, or even a simple
but rigorous definition of the field itself. Even today, it is not uncommon to hear someone offer, in a formal lecture, that artificial intelligence is difficult to define, followed by absolutely no attempt to define it, followed by some interesting research on a problem for which a better solution has been found by some method that is then deemed to be artificially intelligent.

When definitions have been offered, they have often left much to be desired. Intelligent machines may manipulate symbols to solve problems, but simple symbol manipulation cannot be the basis for a broadly useful definition of artificial intelligence (cf., Buchanan and Shortliffe, 1985, p. 3). All computers manipulate symbols; at the most rudimentary level these are ones and zeroes. It is possible for people to assign meaning to these ones and zeroes, and combinations of ones and zeroes, but then where is the intelligence? There is no fundamental difference between a person assigning meaning to symbols in a computer program and a person assigning meaning to binary digits manipulated by a calculator. Neither the program nor the calculator has created any symbolic meaning on its own.

Waterman (1986, p. 10) offered that artificial intelligence was “the part of computer science concerned with developing intelligent computer programs.” This tautological statement offers no basis for designing an intelligent machine or program.

Rich (1983, p. 1) offered, “Artificial intelligence (AI) is the study of how to make computers do things at which, at the moment, people are better,” which was echoed even as recently as 1999 by Lenat (in Moody, 1999). But this definition, if regarded statically, precludes the very existence of artificial intelligence. Once a computer program exceeds the capabilities of a human, the program is no longer in the domain of AI.

Russell (quoted in Ubiquity, 2004) offered, “An intelligent system is one whose expected utility is the highest that can be achieved by any system with the same computational limitations.” But this definition appears to offer intelligence to a calculator, for there can be no higher expected utility than getting four as the right answer to two plus two.1 It might even extend to a pebble, sitting at equilibrium on a bottom of a pond, with no computational ability whatsoever. It is no wonder that we have not achieved our dreams when our efforts have been defined so poorly.

The majority of definitions of artificial intelligence proffered over decades have relied on comparisons to human behavior. Staugaard (1987, p. 23) attributed a definition to Marvin Minsky—“the science of making machines do things that would require intelligence if done by men”—and suggested that some people define AI as the “mechanization, or duplication, of the human thought process.” Using humans as a benchmark is a common, and I will argue misplaced, theme historically in AI. Charniak and McDermott (1985, p. 6) offered, “Artificial intelligence is the study of mental faculties through the use of computational models,” while Schildt (1987, p. 11) claimed, “An intelligent program is one that exhibits behavior similar to that of a human when confronted with a similar problem. It is not necessary that the program actually solve, or attempt to solve, the problem in the same way that a human would.”

1John Searle, in a television interview on CNN, actually described a calculator as “smart” and “intelligent” but contrasted those properties with human psychology (Verjee, 2002).
What then if there were no humans? What if humans had never evolved? Would this preclude the possibility of intelligent machines? What about intelligent machines on other planets? Is this precluded because no humans reside on other planets? Humans are intelligent, but they are only one example of intelligence, which must be defined properly in order to engage in a meaningful discourse about the possibility of creating intelligent machines, be they based in silicon or carbon. I will return to this point later in this chapter.

The pressing question, “What is AI?” would become mere semantics, nothing more than word games, if only the answers did not suggest or imply radically different avenues of research, each with its own goals. Minsky (1991) wrote, “Some researchers simply want machines to do the various sorts of things that people call intelligent. Others hope to understand what enables people to do such things. Still other researchers want to simplify programming.” That artificial intelligence is an extremely fragmented collection of endeavors is as true today as it was in 1991. Yet the vision of what is to be created remains prominent today, even as it did when Minsky (1991) wrote: “Why can’t we build, once and for all, machines that grow and improve themselves by learning from experience? Why can’t we simply explain what we want, and then let our machines do experiments or read some books or go to school, the sorts of things that people do. Our machines today do no such things.”

The disappointing reality is that, actually, even in 1991 machines did indeed do many of these things and the methods that allowed these machines to achieve these results have a long history. What is more disappointing is that this history is mostly unknown by those who work in what they describe as “artificial intelligence.” One of the reasons that less progress has been made than was envisioned in the 1950s stems from a general lack of awareness of the progress that has in fact been made, a symptom that is characteristic of new fields and particularly of AI research. This text seeks to provide a focused explication of particular methods that indeed allow machines to improve themselves by learning from experience and to explain the fundamental theoretical and practical considerations of applying them to problems of machine learning. To begin this explication, the discussion first goes back to the Turing Test.

1.2 THE TURING TEST

Turing (1950) considered the question, “Can machines think?” Rather than define the terms “machines” or “think,” Turing proposed a test that begins with three people: a man (A), a woman (B), and an interrogator (C). The interrogator is to be separated from both A and B, say, in a closed room (Figure 1-1) but may ask questions of both A and B. The interrogator’s objective is to determine which (A or B) is the woman and, by consequence, which is the man. It is A’s objective to cause C to make an incorrect identification. Turing provided the following example of a question posed to the man:

“C: Will X [C’s name for A] please tell me the length of his or her hair?”
“A: My hair is shingled, and the longest strands are about nine inches long.”
Player A may be deceitful, if he so desires. In contrast, the object for B is to help the interrogator. Turing suggested that the probable best strategy for her is to give truthful answers. In order that the pitch of the voice or other external clues may not aid in C’s decision, a teleprinter was to be used for communication between the rooms.

Turing then replaced the original question, “Can machines think?” with the following: “We now ask the question, ‘What will happen when a machine takes the part of A in this game?’ Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman.” This question separates the physical and intellectual capabilities of humans. The form of interrogation prevents C from using sensory information regarding A’s or B’s physical characteristics. Presumably, if the interrogator were able to show no increased ability to decide between A and B when the machine was playing as opposed to when the man was playing, then the machine would be declared to have passed the test. Whether or not the machine should then be judged capable of thinking was left unanswered. Turing in fact dismissed this original question as being “too meaningless to deserve discussion.”
There is a common misconception that the Turing Test involves a machine fooling an interrogator into believing that it is a person. Note from the above description that this is not the essence of the test. The test determines whether or not a machine can be as effective as a man in fooling an interrogator into believing that it is a woman. Since the advent of the Internet and instant messaging, we have seen that it is quite easy for a man to fool an interrogator into believing that he is a woman. Turing quite likely did not envision the challenge to be quite so great.

Turing limited the possible machines to be the set of all digital computers. He indicated through considerable analysis that these machines are universal, that is, all computable processes can be executed by such a machine. Thus, the restriction to digital computers was not a significant limitation of the test. With respect to the suitability of the test itself, Turing thought the game might be weighted “too heavily against the machine. If the man were to try to pretend to be the machine he would clearly make a very poor showing.” Hofstadter (1985, pp. 514–520) related an amusing counterexample in which he was fooled temporarily in such a manner, but note that this obverse version of the Turing Test is not a proper analog because, properly, the man would have to do as well as a woman in pretending to be a machine, and then what would this test be intended to judge?

Turing (1950) considered and rejected a number of objections to the plausibility of a “thinking machine,” although somewhat remarkably he felt that an argument supporting the existence of extrasensory perception in humans was the most compelling of all objections. The “Lady Lovelace” objection (Countess of Lovelace, 1842), referring to a memoir by the Countess of Lovelace on Babbage’s Analytical Engine, is the most common present refutation of a thinking machine. The argument asserts that a computer can only do what it is programmed to do and, therefore, will never be capable of generating anything new. Turing countered this argument by equating it with a statement that a machine can never take us by surprise, but he noted that machines often act in unexpected ways because the entire determining set of initial conditions of the machine is generally unknown: An accurate prediction of all possible behavior of the mechanism is impossible.

Moreover, Turing suggested that a thinking machine should be a learning machine, capable of altering its own configuration through a series of rewards and punishments. Thus, it could modify its own programming and generate unexpected behavior. He speculated that “in about fifty years’ time it will be possible to programme computers, with a storage capacity of about $10^9$ [bits], to make them play the imitation game so well that an average interrogator will not have more than a 70 percent chance of making the right identification after five minutes of questioning” (Turing, 1950). It is now a few years past the time frame of Turing’s prognostication and there is nothing to suggest that we are close to creating a machine that could pass his test.

1.3 SIMULATION OF HUMAN EXPERTISE

The acceptance of the Turing Test focused attention on mimicking human behavior. At the time (1950), it was beyond any reasonable consideration that a computer
could pass the Turing Test. Rather than focus on imitating human behavior in con-
versation, attention was turned to more limited domains of interest. Simple two-per-
son games of strategy were selected. These games received attention for at least 
three reasons: (1) Their rules are static, known to both players, and easy to express 
in a computer program; (2) they are examples from a class of problems concerned 
with reasoning about actions; and (3) the ability of a game-playing computer can be 
measured against human experts.

The majority of research in game playing has been aimed at the development of 
heuristics that can be applied to two-person, zero-sum, nonrandom games of perfect 
information (Jackson, 1985). The term zero-sum indicates that any potential gain to 
one player will be reflected as a corresponding loss to the other player. The term 
nonrandom means that the allocation and positioning of resources in the game (e.g., 
pieces on a chess board) is purely deterministic. Perfect information indicates that 
both players have complete knowledge regarding the disposition of both players’ 
resources (e.g., tic-tac-toe, not poker).

The general protocol was to examine an expert’s decisions during a game so as 
to discover a consistent set of parameters or questions that are evaluated during his 
or her decision-making process. These conditions could then be formulated in an al-
gorithm that is capable of generating behavior that is similar to that of the expert 
when faced with identical situations. It was believed that if a sufficient quantity or 
“coverage” of heuristics could be programmed into the computer, the sheer speed 
and infallible computational ability of the computer would enable it to match or 
even exceed the ability of the human expert.

### 1.3.1 Samuel's Checker Program

One of the earliest efforts along these lines was offered by Samuel (1959), who 
wrote a computer program that learned to play checkers. Checkers was chosen for 
several reasons: (1) There was, and still is, no known algorithm that provides for a 
guaranteed win or draw; (2) the game is well defined, with an obvious goal; (3) the 
rules are fixed and well known; (4) there are human experts who can be consulted 
and against which progress of the program can be tested; and (5) the activity is fa-
miliar to many people. The general procedure of the program was to look ahead a 
few moves at a time and evaluate the resulting board positions.

This evaluation was made with respect to several selected parameters. These pa-
rameters were then included in a linear polynomial with variable coefficients. The 
result of the polynomial indicated the worth of the prospective board under evalua-
tion. The most critical and obvious parameter was the inability for one side or the 
other to move, which signals a loss for that player. This can occur only once in a 
and was tested separately. Another clearly important consideration was the 
relative piece advantage. Kings were given 50% more weight than regular pieces 
(checkers). Samuel tried two alternative methods for including additional param-
ters. Initially, Samuel himself chose these terms, but he later allowed the program to 
make a subset selection from a large list of possible parameters, offered by human 
experts.
To determine a move, the game tree of possible new boards was searched. A minimax procedure was used to discover the best move. The minimax rationale favors making the move that leads to the least damage that the opponent can inflict. The ply, or number of levels to be searched in the tree, was set initially at three, unless the next move was a jump, the last move was a jump, or an exchange offer was possible. Jumps are compulsory in checkers, so extending the search to the point where no jump is possible is termed a search to quiescence. The analysis proceeded backward from the evaluated board position through the tree of possible moves, with the assumption that at each move the opponent would always attempt to minimize the machine’s score, whereas the machine would act to maximize its score. Under these conditions, the search was continued until these circumstances were no longer encountered or until a maximum of 20 levels had been searched.

After initial experimentation in which the selected polynomial had four terms (piece advantage, denial of occupancy, mobility, and a hybrid third term that combined control of the center and piece advancement), the program was allowed to select a subset of 16 parameters from a list of 38 chosen parameters. Samuel allowed the computer to compete against itself; one version, Alpha, constantly modified the coefficients and parameters of its polynomial, and the other version, Beta, remained fixed (i.e., it was replaced by Alpha after a loss). A record of the correlation existing between the signs of the individual term contributions in the initial scoring polynomial and the sign of the change between the scores was maintained, along with the number of times that each particular term with a nonzero value was used. The coefficient for the polynomial term of Alpha with the then-largest correlation coefficient was set at a prescribed maximum value with proportionate values determined for all the remaining coefficients. Samuel noted some possible instabilities with this modification technique and developed heuristic solutions to overcome these problems. Term replacement was made when a particular parameter had the lowest correlation eight times. Upon reaching this arbitrary limit, it was placed at the bottom of the reserve list and the first parameter in the reserve list was inserted into the scoring polynomial.

After a series of 28 games, Samuel described the program as being a better-than-average player. “A detailed analysis of these results indicated that the learning procedure did work and that the rate of learning was surprisingly high, but that the learning was quite erratic and none too stable” (Samuel, 1959). In retrospect, the correctness of this analysis can be doubted, as will be discussed shortly.

In 1962, at the request of Edward Feigenbaum and Julian Feldman, Samuel arranged for a match between his program and Robert W. Nealey, a purported former Connecticut state checkers champion. Samuel’s program defeated Nealey, who commented (cited in Samuel, 1963):

Our game . . . did have its points. Up to the 31st move, all of our play had been previously published, except where I evaded “the book” several times in a vain effort to throw the computer’s timing off. At the 32-27 [a specific move] loser and onwards, all of the play is original with us, as far as I have been able to find. It is very interesting to me to note that the computer had to make several star moves in order to get the win,
and that I had several opportunities to draw otherwise. That is why I kept the game going. The machine, therefore, played a perfect ending without one misstep. In the matter of the end game, I have not had such competition from any human being since 1954, when I lost my last game.


Unfortunately, far more acclaim was given to this result than was deserved. Schaeffer (1996, p. 94) indicated that Nealey was in fact not a former Connecticut state champion at the time of the match again Samuel’s program, although he did earn that title in 1966, four years later. Nealey did not enter the U.S. Championship Checkers Tournament and, thus, the strength of his play at the national level was based more on opinion than on record. Schaeffer (1996, pp. 94–95) reviewed the sequence of moves from the Nealey match, and with the aid of Chinook (the current world champion checkers program designed by Schaeffer and his colleagues), indicated that Nealey made several blunders in the game and that Samuel’s program also did not capitalize on possible opportunities. In sum, the glowing description that Nealey gave of Samuel’s program’s end game is well accepted in common literature, but is an overstatement of the program’s ability.

In 1966, Samuel’s program was played against the two persons vying for the world championship. Four games were played against each opponent, with Samuel’s program losing all eight games (Schaeffer, 1996, p. 97). As described in Fogel (2002), Samuel’s program also lost to other checkers programs in the 1970s, and was characterized in 1977 by checkers authority Richard Fortman as being not as good as even a Class B player, which rates in succession below Class A, Expert, Master, and Grand Master. Samuel’s concept for a learning machine was pioneering, and generally faithful to the original dream of a computer that could teach itself to win. The results, however, were not particularly successful.

1.3.2 Chess Programs

Researchers in artificial intelligence have also been concerned with developing chess programs. Initial considerations of making machines play chess date to Charles Babbage (1792–1871). Babbage had described the Analytical Engine, a theoretic mechanical device that was a digital computer, although not electronic. This machine was never built, but an earlier design, the Difference Engine, was in fact constructed successfully in 1991 (Swade, 1993). Babbage recognized that, in principle, his Analytical Engine was capable of playing games such as checkers and chess by looking forward to possible alternative outcomes based on current potential moves.

Shannon (1950) was one of the first researchers to propose a computer program to play chess. He, like Samuel later, chose to have an evaluation function such that a program could assess the relative worth of different configurations of pieces on the board. The notion of an evaluation function has been an integral component of every chess program ever since. The suggested parameters included material advantage, pawn formation, positions of pieces, mobility, commitments, attacks and
options (see Levy and Newborn, 1991, pp. 27–28). Shannon noted that the best move can be found in at least two ways, although the methods may be combined: (1) Search to a given number of moves ahead and then use a minimax algorithm, or (2) selectively search different branches of the game tree to different levels (i.e., moves ahead). The second method offers the advantage of preventing the machine from wasting time searching down branches in which one or more bad moves must be made. This method, later termed the alpha-beta algorithm, has been incorporated in almost every current chess playing program.

Turing (1953) is credited with writing the first algorithm for automatic chess play. He never completed programming the procedure on a computer but was able to play at least one game by hand simulation. Turing’s evaluation function included parameters of mobility, piece safety, castling, pawn position, and checks and mate threats. The one recorded game (see Levy and Newborn, 1991, pp. 35–38) used a search depth of two ply and then continued down prospective branches until “dead” positions (e.g., mate or the capture of an undefeated piece) were reached. In this game, the algorithm was played against a presumed weak human opponent (Levy and Newborn, 1991, p. 35) and subsequently lost. Turing attributed the weakness of the program to its “caricature of his own play” (Levy and Newborn, 1991, p. 38).

The first documented working chess program was created in 1956 at Los Alamos. An unconfirmed account of a running program in the Soviet Union was reported earlier by Pravda (Levy and Newborn, 1991, p. 39). Shortly thereafter, Bernstein et al. (1958) described their computer program, which played a fair opening game but weak middle game because the program only searched to a depth of four ply. Newell et al. (1958) were the first to use the alpha-beta algorithms (Shannon, 1950). Greenblatt et al. (1967) are credited with creating the first program, called Machack VI, to beat a human in tournament play. The program was made an honorary member of the United States Chess Federation, receiving their rating of 1640 (in Class B, which ranges from 1600 to 1800). Machack VI used a search of at least nine ply.

In 1978, Chess 4.7, a revised version of a program written originally by Atkin, Gorlen, and Slate of Northwestern University, defeated David Levy, Scottish chess champion, in a tournament game. Levy was “attempting to beat the program at its own game,” and returned in the next match to a “no nonsense approach,” presumably to win (Levy and Newborn, 1991, p. 98, 100). Belle, written by Thompson and Condon, was the first program that qualified, in 1983, for the title of U.S. Master.

In the 1980s, efforts were directed at making application-specific hardware that facilitated searching large numbers of possible boards and quickly calculating appropriate evaluations. Berliner created Hitech, a 64-processor system. Hsu produced an even more powerful chip and its resident program, now known as Deep Thought, quickly outperformed Hitech. Deep Thought was able to search to a level of 10 ply and became the first program to defeat a world-class grand master, Bent Larsen. In 1989, Deep Thought, then rated at 2745, played a four-game match against David Levy. Levy admitted, “It was the first time that [I] had ever played a program rated higher than [I] was at [my] best” (Levy and Newborn, 1991, p. 127) and predicted correctly that the machine would win 4 to 0. In 1990, Anatoly
Karpov, the former world champion, lost a game to a Mephisto chess computer while giving a simultaneous exhibition against 24 opponents.

The pinnacle of beating a human world champion in match play was achieved finally in May 1997 when IBM’s Deep Blue, the successor to Deep Thought, defeated Garry Kasparov, scoring two wins, one loss, and three draws. The previous year, Kasparov had defeated Deep Blue, scoring three wins, one loss, and two draws. The computing horsepower behind Deep Blue included 32 parallel processors and 512 custom chess ASICs, which allowed a search of 200 million chess positions per second (Hoan, cited in Clark, 1997). Although the event received wide media attention and speculation that computers had become “smarter than humans,” surprisingly little attention was given to the event in the scientific literature. McCarthy (1997) offered that Deep Blue was really “a measure of our limited understanding of the principle of artificial intelligence (AI) ... this level of play requires many millions of times as much computing as a human chess player does.” Indeed, there was no automatic learning involved in Deep Blue, although some attempts had been made to include methods of adjusting coefficients in a polynomial evaluation function but these were not incorporated into the final product (Fogel, 2002). A. Joseph Hoan, Jr., a member of the team that developed Deep Blue, remarked (in Clark, 1997): “we spent the whole year with chess grand master, Joel Benjamin, basically letting him beat up Deep Blue—making it make mistakes and fixing all those mistakes. That process may sound a little clunky, but we never found a good way to make automatic tuning work.” Between games, adjustments were made to Deep Blue based on Kasparov’s play, but these again were made by the humans who developed Deep Blue, not by the program itself.

Most disappointingly, IBM decided to disassemble Deep Blue after its historic win over Kasparov. This not only prevented a rematch, but stifled further study of the program and machinery, making the result irreproducible. The result of the 1997 match with Kasparov was limited therefore to a “proof of existence”—it is possible for a machine to defeat the human world champion in chess—but its contribution to the advancement of computer-based chess melted away along with its hardware.

Judging by the nearly linear improvement in the United States Chess Federation rating of chess programs since the 1960s (Levy and Newborn, 1991, p. 6), which continues to this day with programs such as Fritz, Shredder, Deep Junior, and others, the efforts of researchers to program computers to play chess must be regarded as highly successful. But there is a legitimate question as to whether or not these programs are rightly described as intelligent. The linear improvement in computer ratings has come almost exclusively from faster computers, not better chess knowledge, and certainly not from machines teaching themselves to play chess. Schank (1984, p. 30) commented, over a decade before Deep Blue’s victory, “The moment people succeeded in writing good chess programs, they began to wonder whether or not they had really created a piece of Artificial Intelligence. The programs played chess well because they could make complex calculations with extraordinary speed, not because they knew the kinds of things that human chess masters know about chess.” Thirteen years later, after Deep Blue’s victory, Schank’s comments were just as salient. Simply making machines do things that people would describe as requiring intelli-