1 First Things

Anyone teaching a course [on AI] ... will have to decide what artificial intelligence is, even if only because inquiring minds want to know.
—Stuart Russell and Eric Wefald (1991)

"Can machines think?" [is] as ill-posed and uninteresting as "Can submarines swim?"
—Edsger W. Dijkstra (ca 1970)

Our minds contain processes
that enable us to solve problems we consider difficult.
"Intelligence" is our name
for whichever of those processes we don't yet understand
—Marvin Minsky (1985)

Introduction

From golems to androids, manmade intelligences have been a dream and nightmare of mankind for centuries. In the 1950s, electronic brains led to the birth of the science of artificial intelligence. Will AI, as the field is commonly called, fulfill its promise to convert mankind's fantasies into reality?

We'll begin exploring the nature of AI by examining its goals, tools, and accomplishments, and some of the debates it has engendered. Such an examination should give us a revealing picture of the current state of this promising discipline.

Next, we'll discuss the why's and wherefore's of this text. Why the emphasis on mathematics? What do future chapters hold in store?

The final sections introduce two important topics: the computation problem and expert systems. The computation problem permeates AI, but is not

always evident. Meeting it face to face now is important because overlooking its presence is a ticket to disaster. Expert systems provide a unified way of viewing most of AI.

1.1 Delimiting AI

We'll examine AI from three different viewpoints, or "coordinates":

$$\begin{array}{l} \text{goals,} \\ \text{methods or tools and} \\ \text{achievements and failures.} \end{array} \tag{1.1}$$

For example, your goal may be to understand what AI is all about; your method, talking to AI researchers; and your achievement, a new overview of AI. Some parts of AI (such as machine learning) are primarily defined by goals, others (such as neural networks) primarily by methods. Achievements and failures give information on how a field has progressed.

Some Goals of AI

When you read the following list, interpret words like "reasoning" and "understanding" as referring to the *results*, not the *methods*. In other words, focus on a program's output rather than its algorithm. (Mimicking human algorithms is a concern of cognitive science, not AI.)

- **Reasoning**: Given some general knowledge together with some specific facts, deduce certain consequences. For example, given knowledge about diseases and symptoms, diagnose a particular case on the basis of information about the symptoms. The most difficult type of reasoning is based on what people call "common sense."
- **Planning**: Given (a) some knowledge, (b) the present situation, and (c) a desired goal, decide how to reach the goal; that is, use *goal-directed reasoning*. How do planning and reasoning differ? They overlap, but, roughly, reasoning seeks the answer to What?; planning seeks the answer to How? For example, "What sort of student am I?" versus "How can I be an A student?"
- **Learning**: Acquiring knowledge (learning) is a central issue since knowledge must be acquired before it can be used. In some situations, it is feasible to build knowledge into a system. In others, it is infeasible or undesirable. Then we want a system that can repeatedly extend its knowledge base in a coherent fashion by acquiring new facts and integrating

them with previous knowledge, often by some process of abstraction. For example, if a system is exposed to various examples of chairs, how can it abstract the concept "chair"?

The previous goals are rather general in nature and are relevant to many parts of AI. We now look at goals that may be viewed as more specific. Although they draw on results in the previous areas, they are very much separate parts of AI with their own tools and problems.

- Language Understanding and Use: Obviously, this relies heavily on reasoning and learning, but it deserves a separate category. "Common sense" plays an important role in language. Unfortunately, common sense is an extremely elusive topic that appears to require a considerable knowledge base. Attempts to understand spoken language must face additional complications.
- Processing Visual Input: Vision is only one type of sensory input that must be processed, but it is by far the most complex. Abstracting useful information from visual input is proving very difficult.
- Robotics: Robotics must marry AI with engineering. In all but the simplest industrial settings, reality is dauntingly complex. The AI techniques used in robotics must produce results in real time and, for an autonomous robot, must not require excessive computer power.

Some Tools of AI

Knowledge about knowledge is the focus of AI. Knowledge is given either *declaratively*—in declarative statements—or *procedurally*—by procedural rules. Specific knowledge tends to be represented declaratively and general knowledge procedurally. (The situation is not this cut-and-dried, but the distinction is still useful.) Declarative knowledge is stored in what is called a *knowledge base*. Knowledge about knowledge provides tools for interacting with the knowledge base:

- Knowledge Organization Tools: Data structures and algorithms facilitating the organization of the knowledge base.
- Knowledge Manipulation Tools: Methods for extracting new knowledge from the knowledge base; for example, reasoning and planning.
- Knowledge Acquisition Tools: Methods for incorporating new knowledge into the knowledge base or modifying the tools ("learning"). The border between acquisition and manipulation is fuzzy.

In many areas of computer science, algorithms are primary and data structures are secondary. In contrast, knowledge representation is a central problem in AI. The form of declarative knowledge (the data structures) limits what

we can state and the ease with which we can manipulate it. AI's declarative knowledge is seldom considered "just data." Thus, the tools of AI could be thought of in terms of what can be incorporated in the data structures. Here are some data structures and where to find them and their tools.

- Limited Structure: Relatively unstructured search spaces are attractive because they impose few restrictions. Sadly, the lack of structure makes computations overwhelming for all but the simplest problems (Chapters 2 and 13).
- Mathematical Logic: Mathematical logic allows us to represent facts about the world in a form that can be manipulated (Chapters 3–6).
- Logic-like Representations: Representational awkwardness and other handicaps motivated some researchers to seek alternatives. Some approaches, such as rule-based systems and semantic nets, can be recast in the framework of logic (Chapter 6). Other approaches, such as reasoning by analogy as in case-based reasoning, use other methods and are lightly touched upon in Chapter 14.
- Numerical Information: Numerical information can play a central role in describing uncertainty about the world, as in "a 40% chance of rain" (Chapters 8 and 9).
- Nonsymbolic Structures: The previous structures are designed to represent and manipulate information symbolically. A growing number of researchers have questioned this approach (Chapters 10, 11, and 13).

On another level, we could say that the tools of AI are those things that provide the basis for creating the knowledge tools. They tend to fall into four areas:

- Hardware: AI makes heavy use of computers for developing and testing ideas. Some parts can benefit from special-purpose devices.
- Software: AI's large software systems make it an important developer and consumer of programming tools.
- Mathematics: Some parts of mathematics have proven useful in AI. (Every formal manipulation of concepts is a part of mathematics.)
- Heuristics: Sometimes called "rules of thumb," heuristics are empirical principles. Heuristics may use mathematics, but often do not.

What Has AI Given the World?

Many of AI's contributions contain no AI: They are simply tools that were developed to aid AI research. We'll begin with these spinoffs and move on to results that do contain some AI. There is no consensus on where to draw a line between contributions containing little or no AI and those with significant amounts. Many draw the line just before or just after "game-playing programs."

- Timesharing: Much early work on timesharing was done by MIT's project MAC—a dual acronym meaning either "machine-aided cognition" or "multiple-access computing." (Some wags called it "man against computer.")
- Windows and Graphical User Interfaces: These were developed at Xerox's Palo Alto Research Center to provide easier computer access for AI researchers.
- Programming Paradigms: These include
  - constraint propagation (now used in spreadsheet programs),
  - object-oriented programming,
  - functional programming (the basis of Lisp), and
  - logic, or declarative, programming (the basis of Prolog).
- Fuzzy Controllers: "Fuzzy logic" leads to more stable and flexible means of regulating machines.
- Game-Playing Programs: Game playing was a favorite topic in the early years of AI research. By 1995, the best artificial chess player could beat all but the best human players. Backgammon programs achieved a similar level: One beat the world champion because lucky rolls of the dice compensated for somewhat inferior play.
- Expert Systems: Commercial expert systems have been proliferating in recent years and many businesses are using special-purpose software to write expert systems for in-house use.
- Natural Language Interfaces: A limited ability to understand natural language is providing friendlier user interfaces for some programs.
- Dictation Systems: Systems able to transcribe speech have begun to appear on the market. So far, vocabulary and speed are rather limited.

The flip side of achievement is failure—the skeleton in the closet. Here are three of them.

- Wild Optimism: The seeds of a variety of failures were planted in the 1950s—the early, heady years of AI when almost everything was "just around the corner." In the 1980s, a minor relapse into unbridled optimism was caused by the rebirth of neural networks—a methodology inspired

by the highly interconnected, self-modifying nature of biological neural systems.

- Game-Playing Programs: Many hoped that studying games would lead to significant progress in AI. The rate of return has been low, however—perhaps because competitions tend to focus on immediate improvement rather than new ideas.
- Ad Hoc Developments: Much research has been based on ad hoc methods rather than solid foundations. People argue about whether seat-of-the-pants design is inherent in the subject matter of AI or just a passing stage. Advocates of ad hoc methods are called *scruffies*; advocates of theoretical methods are called *neats*.

Results versus Methods: Cognitive Science

Artificial intelligence is an invention.
In contrast, a theory of human intellect is a discovery.

—Morton Wagman (1991)

For some, a major goal of AI is the construction of an artificial intelligence having human-level abilities. Progress has certainly been made, but the goal is still far away, if not impossible. Other people are concerned with the methods *humans* use to achieve their abilities. As noted earlier, these people are *cognitive scientists*.

Like AI, cognitive science is an umbrella field related to "intelligence." Cognitive science, which includes topics like cognition and consciousness, seems to be striving to achieve many of the goals listed for AI. Unlike AI, cognitive science focuses on learning how human minds achieve such goals rather than on creating artificial methods for achieving them. There are a variety of introductions to cognitive science; for example [14] and [49].

Allen Newell [32] was a major advocate for developing unified theories of cognition that can be tested and expressed through programs. He argues cogently that both cognitive science and AI will profit from such attempts in the short term, but will go their separate ways in the long term [32, p. 57]. To see why this might be so, consider a crude parallel—a slightly fictional history of flight. Cognitive science corresponds to understanding bird flight, and AI to creating artificial flight. Understanding and adapting some aspects of bird flight informed the early development of artificial flight. Conversely, attempts at artificial flight provided tests for the understanding of bird flight. Major progress required an understanding of the principles of aerodynamics, at which point the methods employed by birds were no longer relevant. (Actually, studying flying fish may have been more productive for early attempts at flying.)

Exercises

1.1.A. What are some goals of AI?

1.1.B. What are some general tools of AI?

1.1.C. What are some contributions of AI?

1.1.D. What is the difference between AI and cognitive science?

1.2 Debates

There is nothing which is not the subject of debate, and in which men of learning are not of contrary opinions. The most trivial question escapes not our controversy, and in the most momentous we are not able to give any certain decision.

—David Hume (1740)

Consciousness is a subject about which there is little consensus, even as to what the problem is. Without a few initial prejudices one cannot get anywhere.

—Francis Crick (1994)

One of the ongoing debats in AI is the definition of the AI field itself. Actually, the variety in the field probably makes it impossible to give a concise definition that is neither too broad nor too narrow. To see why this is so, try the much simpler problem of defining what is meant by a sport. Your definition should include bowling and recreational cycling, but not chess or dancing.

Here are three debates that provide some insights about AI.

Consciousness and Intelligence

A better understanding of cognitive science topics like intelligence and consciousness could benefit AI research. Thus we'll look briefly at these debates, even though they do not belong in AI.

A question like "Can machines think?" is difficult. We often start from the premise that we understand what this question means when, in fact, ongoing debates show that we have not yet figured out what we're talking about. Even the first step—agreeing on the definition of "intelligence"—has not been taken. Some people believe that the most famous proposed test for machine intelligence, the Turing test, should be regarded as a definition of intelligence. Other people disagree. (See Exercise 1.2.1.)

Perhaps thought and intelligence are the wrong issues to address. Instead, consciousness may be a more fundamental issue. We seem to know less about this subject than some experts would like to believe. The study of consciousness belongs to philosophy, psychology, and cognitive science. If this line of study appeals to you, you may find the books by Churchland [8], Dennett [12], and Moody [31] of interest.

The range of beliefs (or hopes) regarding intelligence and consciousness is quite broad.

- At one extreme, strong AI supporters maintain that it is possible to create an intelligent, conscious machine and that something like the Turing test (Exercise 1.2.1) is adequate to determine if the machine is intelligent and conscious. One expression of this is the *physical symbol hypothesis* of Newell and Simon [33]. They define a physical symbol system to be something that is capable of manipulating physical patterns (such as data in a computer or strengths of connections among neurons) and hypothesize that such a system is necessary and sufficient for implementing general intelligent behavior.
- At the other extreme are those who maintain either (a) that intelligent, conscious behavior has a nonphysical component (as in Cartesian dualism) or (b) that it involves something inherently biological. These people conclude machines will never achieve such behavior.

Given the current state of AI, researchers need not worry about such issues any more than the Wright brothers needed to worry about the sound barrier.

Symbols versus Connections

The symbols versus connections debate might also be described as "intelligence by design" versus "intelligence as an emergent property."

The traditional approach to AI has been symbolic; that is, knowledge is represented at a symbolic level comprehensible to us. The strict symbolic viewpoint is that the way to make real progress in AI is through the development of powerful data structures and algorithms for the representation and manipulation of knowledge on a symbolic level. Most defenders of this view believe the symbolic approach mimics conscious human reasoning. The choice of a symbolic framework has been debated. Some want to base the symbolic approach on mathematical logic; others insist that numerical methods should play a central role.

There has recently been a revival of the connectionist approach. Like much of AI, this approach was born amidst the rosy predictions of the 1950s. It nearly disappeared in 1969 after Minsky and Papert [30] emphasized the limitations of the methods then available. Interest blossomed anew in the 1980s. Since then, considerable research has been done using simulations of

networks of simple interconnected processors, that is, *neural networks.* Strict connectionists believe that one should design complex networks of simple processors and then train these networks. Intelligence, they maintain, will emerge as a consequence, but won't be found in the parts of the network separately. This is the sort of internal representation and manipulation of knowledge that the human brain apparently uses on the physiological level, with neurons as processors.

Which approach is better? The answer may depend on the application. It may be best to combine the approaches—people are experimenting with hybrid systems. At any rate, it's too soon to tell.

The Role of Theory

The word "theory" encompasses mathematics as well as such things as the theory of general relativity. It does not include simple facts and rules of thumb based on them. For example, the commonsense advices "get a good night's sleep before an exam" is not a theory. It's a heuristic rule based on personal observation. What, then, is the practical relevance of theory for AI?

"The theory *is* the program" view of some nontheorists is at one extreme. This attitude should not be confused with the idea that computer programs in AI (should) play the role of experiments—no one claims that a theory is an experiment. In contrast, "The theory is the program" means you may ask how well the program works but you can't ask for a foundation on which the program is based.

At the other extreme is the ultra-logicist claim that, ultimately, AI will succeed by employing a theoretically justified system of symbolic reasoning.

Naturally, most researchers' beliefs lie between these two extremes. The issue then is "What is the best blend between heuristics and theory?" The answer to this question depends on the researcher, on the subject, and on its state of development: On the researcher, because abilities vary from person to person; on the subject, because simpler areas are more easily fit into a theoretical framework; and on the state of development because mathematics is gradually making greater inroads into various areas of AI.

*Exercises

To the student: These exercises are likely to be time-consuming. Most instructors (myself included) won't assign any because of time pressure. Read them anyway—they provide food for thought.

To the instructor: See above.

1.2.1. The *Turing test* [51]: An evaluator E is allowed access to two subjects W (a woman) and X (not a woman) only through a remote terminal. The experimenter tells E that exactly one of W and X is a woman and instructs E to determine which it is by whatever method E wishes using the remote terminal. Each of W and X attempts to react like a woman when responding to E's questions. If E decides that X is a woman, then E has been deceived. By averaging over many E's, W's, and X's, we can obtain a success rate for deception. In particular, we can compute the deception rate when X is a man—the deception rate for men. We can also compute the deception rate for a computer program. In the Turing test, the program is declared to possess intelligence if its deception rate is at least as great as the deception rate for men.

(a) Consider the following statement: "The Turing test is based on the idea that the ability to misrepresent oneself is a measure of intelligence." Do you agree? Why? If you agree with it, do you think that ability is a measure of intelligence? Why?

(b) In some statements of the Turing test, the deception rate for a computer program is simply required to exceed some value. Which version do you think is better? Why?

(c) Suppose a species as intelligent as humans were found. (Intelligence in this sentence does not refer to the Turing test, but to a "commonsense" assessment.) Do you think such aliens could pass the Turing test? Why?

(d) Given the existence of an intelligent alien species, suggest and defend a less species-biased test for computer intelligence.

(e) As stated, passing the Turing test depends on the computer's possessing extensive knowledge of the nature of human beings, both physical and psychological, as well as their culture, history, literature, and so forth. Suggest and defend modifications of the Turing test that would reduce the need for such knowledge. To what extent can such a need be eliminated without affecting the validity of the test?

(f) More generally, can you formulate a better test?

1.2.2. Suppose we are considering cognitive skills, learning abilities, or some other human skill that is relevant to AI. Imagine a three-sided debate:

1. The (nearly) best way to achieve this skill has been found by evolution.
2. By reason and experiment, we'll be able to improve considerably on human skills.
3. Neither of the two previous views is correct.

Come to class prepared to carry out such a debate. (You may be assigned a particular viewpoint to defend.)

1.2.3. For each of the three sides of the debate in the previous exercise, describe the implications for AI work on a particular skill if the side is correct.

1.2.4. Newell [34, p. 19] lists a variety of things a mind is able to do, many of which are reproduced below. Which of these abilities do you think a computer program should have in order to deserve being considered a major AI project? Explain your choices.
Hint. There is a wide latitude for acceptable answers, but you may have to decide what *you mean by AI* in order to answer.

(a) Behave flexibly as a function of the environment

(b) Exhibit adaptive (rational, goal-oriented) behavior

(c) Operate in real time

(d) Operate in a rich, complex, detailed environment

- Perceive an immense amount of changing detail
- Use vast amounts of knowledge
- Control a motor system of many degrees of freedom

(e) Use symbols and abstractions

(f) Use language, both natural and artificial

(g) Learn from the environment and from experience

(h) Acquire capabilities through development

(i) Operate autonomously, but within a social community

(j) Be self-aware and have a sense of self

1.3 About This Text

The paradox is now fully established that the utmost abstractions are the true weapons with which to control our thought of concrete fact.

—Alfred North Whitehead (1925)

Understanding in mathematics cannot be transmitted by painless entertainment any more than education in music can be brought by the most brilliant journalism to those who have never listened intensively. Actual contact with the content of living mathematics is necessary.

—Richard Courant (1941)

Teach nothing that pupils can teach themselves.

—Amos Bronson Alcott (1799–1888)

Mathematics is the term we use to describe the process of symbolically deducing conclusions from conceptual assumptions, whether these be the axioms of

geometry, the laws of physics, or the assumptions in economics' utility theory. Mathematics with bad assumptions is useless; with good assumptions, it is a wonderful tool.

Heuristics is the term we use to describe empirical principles and techniques, such as "Avoid the use of GOTO," "The best offense is a good defense," and "graphical user interfaces." Good heuristics whose limits are well understood are very useful.

Programming is a means of testing ideas, creating tools, and generating information that may spark new research. Because of AI's complexity, we often use special languages (most notably Lisp and Prolog) or simulator packages (especially for neural nets).

Programming, heuristics, and mathematics are all important in AI.

Because AI is a large field, textbook authors must make choices. Most authors emphasize heuristics and relatively simple programming exercises. Since writing large programs and studying mathematics are time-consuming, this approach allows the broadest coverage of topics. After learning some Lisp or Prolog and taking a course that involves a large programming project, you should be able to study AI programming methods and write such programs. On the other hand, it's much harder to study mathematics on your own.

My goal is to provide an introductory AI course based on the most important mathematics and its applications. To keep the length manageable, material must be cut. My algorithm is simple: Focus on important AI topics that involve the most broadly applicable mathematics and cut back on others. What does that leave? The main mathematical tools for representing and manipulating knowledge symbolically are (a) various forms of logic for qualitative knowledge and (b) probability and related concepts for quantitative knowledge. The main tools for manipulating knowledge nonsymbolically, as in neural nets, are optimization methods and statistics. I've organized that material as follows.

- Trees and Search: Since search plays a central role in AI, elementary aspects of search trees are discussed in Chapter 2. Some additional aspects of search are briefly discussed in Chapter 13 after the necessary probability theory has been introduced in Chapter 12.
- Classical Mathematical Logic: First-order predicate logic, the starting point for the use of logic in AI, is presented in Chapters 3 and 4. Prolog is introduced to show how the concepts and results can be implemented in a programming language. The reasoning engine in Prolog combines a search strategy with a deductive method from logic. (You won't, however, learn how to program in Prolog from this brief introduction.)
- Uncertainty in Reasoning: AI systems based on classical mathematical logic have various shortcomings. Among these are the following:
  - We can't easily allow for general rules that have exceptions (for example, "mammals have legs" and "whales are mammals without legs").

- We can't allow for uncertain statements (for example, "When the barometer is falling, it often rains by the following day.")

Qualitative approaches based primarily on extending logic are discussed in Chapter 6. Quantitative approaches are discussed in Chapters 8 and 9 after the necessary probability theory has been introduced in Chapter 7.

- Automatic Classification: An alternative to incorporating knowledge-based rules into expert systems is to design programs that develop their own "rules" from examples. These are called *pattern classifiers* and are discussed in Chapters 10, 11, and 13. After discussing neural nets and optimization in Chapter 11, I digress to introduce some probability, statistics, and information theory in Chapter 12. This is applied to neural nets and decision trees in Chapter 13.
- Other Things: The previous material omits important areas of AI. One is robotics, in which sensory-input processing (especially vision) and motion planning involve considerable mathematics. Another is language, where linguistics and speech processing use mathematics. The final chapter contains brief introductions to these omissions and to some less mathematical topics so that you'll have a bit of background and some references for further study.

1.4 The Computation Problem in AI

Any program that will successfully model even a small part of intelligence will be inherently massive and complex. Consequently artificial intelligence continually confronts the limits of modern computer-science technology.

—J. Michael Brady, Daniel G. Bobrow, and Randall Davis (1993)

Designing algorithms is a central problem in almost any computer oriented field, and AI is no exception. Unfortunately, algorithms are particularly troublesome in AI. Three reasons for this are as follows.

- Complexity: Problem complexity makes designing and implementing algorithms difficult.
- Time: Algorithms frequently explore potential solutions in the course of searching for an acceptable one. For problems of realistic size, a simple search process may take too long because of *combinatorial explosion*—a rapid growth in the number of possible solutions. Unfortunately for algorithm design in AI,

Very rapid growth is typical in AI problems.

- Impossibility: It may be impossible to design an algorithm for the given problem. Here's a specific example. We would like to design an algorithm that takes as input (a) a computer program in some suitable language and (b) some data for the program. The algorithm must determine if the program will stop or run endlessly—the *halting problem.* In designing the algorithm, we imagine an abstract computer having infinite storage. (Of course, compromises will have to be made when we get around to implementing the algorithm.) In a classic paper in 1935, Turing *proved* that *no such algorithm can exist.* Thus, the problem is impossible.

Here are some ways of dealing with these problems.

- Find a much better algorithm: This is the ideal solution. Unfortunately, we often cannot find a much better algorithm.
- Settle for an algorithm that sometimes fails: These are algorithms that sometimes fail either by stopping with no solution or, worse, by giving an incorrect solution. It's possible to create such an algorithm by imposing a time limit on another algorithm. For example, the famous simplex algorithm in linear programming has a very bad worst-case time and a very good average-case time [6]. Thus, an intelligently designed time limit would lead to a solution in most cases. For this approach to be useful, we must know from theory or experience that failure is relatively rare.
- Settle for an approximate solution: Such a solution is often good enough. Simon coined the term *satisficing* for finding a good enough solution. Sometimes, obtaining good approximate solutions may be as difficult as the original problem.
- Replace the problem with an easier one: Solving the easier problem may produce useful results. Also, exploring the easier problem may lead to ideas for the original problem.
- Give up: No comment.

All of these approaches are used in AI, often in combination. Inventing compromise algorithms is a tricky, creative business. Mathematics may help in inventing and assessing compromises, but is seldom sufficient. The final weighing of gains and losses in a compromise is a value judgment based on your goals.

How much time should an algorithm be allowed to take? More time often means a better result. On the other hand, speed of response is important; for example, a user is less likely to use a sluggish expert system than a quick one. Figure 1.1 illustrates this idea. Unfortunately, the information needed to construct the curves in the figure is seldom available. In this case, an *anytime algorithm* can be quite useful. This is an algorithm that can be interrupted at any time to obtain an approximate answer. Here's a simple example of such an algorithm. Suppose we know that f is continuous on the interval $[a, b]$, that $f(a) < 0$, and that $f(b) > 0$. We want to obtain an estimate for an

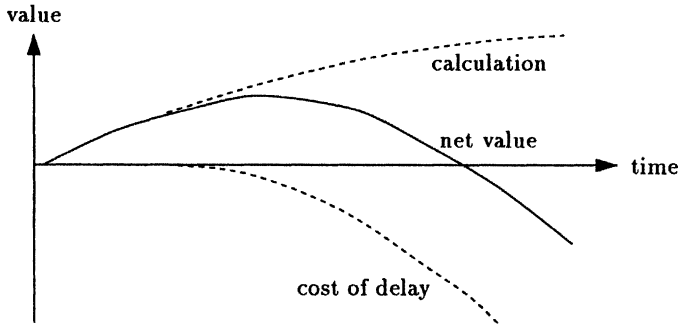

Figure 1.1 The typical effect of response time. The vertical scale measures value in some unspecified manner. The upper dashed curve shows how the value of a result varies with computation time. It ignores the costs of time delay. The lower dashed curve shows the cost of delay in response time. The middle curve, which combines the two, shows the net value of the response. Computation should end at the middle curve's maximum: Even though more calculation would give a better result, the cost of delay outweighs the gain.

$x \in (a, b)$ such that $f(x) = 0$. Simply repeat the following two steps: Let $c = (a + b)/2$. If $f(c) \leq 0$, let $a = c$; otherwise, let $b = c$. Whenever the algorithm is interrupted, it returns the estimate c for x.

NP-Hard Problems

Theoretical computer scientists consider an algorithm to be fast if its running time can be bounded by a polynomial in the number of bits needed to express the input and the output. This means that, even in the worst case, the algorithm is reasonably fast on very large problems. It says nothing about average running time. Indeed, it may be very difficult to define an average running time since it may be unclear what to average over.

In the theory of algorithms, a certain class of problems is called *NP-complete*. Hundreds of problems of interest to computer scientists have been shown to be NP-complete. It has been proved that either a fast algorithm exists for *all* NP-complete problems, or no fast algorithm exists for *any* NP-complete problem. Since no fast algorithm has been found after many years of research, it seems unlikely that any exists.

An *NP-hard* problem is one that is at least as difficult as an NP-complete problem. Even when a problem is NP-hard, there may well be an algorithm that works well on the situations that arise in actual usage—that is, the worst cases simply don't arise in practice. (Of course, as soon as you decide this and release your program to the world, Murphy's law dictates that someone will

come up with a use where the worst cases occur.) In other words, the relevant time is the average running time over inputs that will actually occur. Unfortunately, this time is usually difficult or impossible to determine theoretically.

Polynomial time algorithms and NP-complete algorithms are the bottom levels of a whole series of increasingly more difficult problems that are studied in complexity theory. Some AI problems are NP-complete. Many more are even more difficult. As a result, compromises of some sort are often needed.

Aside. Here's a technical note for those who want to know a bit more about NP-complete. Let $|y|$ denote the number of bits needed to describe y. We say that an algorithm is (at most) "g time" if the running time of the algorithm with input x is bounded by $g(|x|)$.

Suppose we want to determine whether certain things in a set S have some property F. This is called a *recognition problem.* A recognition problem is in the class P if there exists a polynomial time algorithm that can determine if $F(x)$ is true or false. For example, S could be the positive integers and F could be "composite" (not a prime). In this case, $F(x)$ is true if and only if x is not a prime. No polynomial time algorithm is known for this example.

It may be much easier to verify that $F(x)$ is true for a given x if we're given some additional information. This added information is called a *certificate.* Thus, a certificate could change a hard problem into an easy one. (Of course, it might be *very hard* to create such a certificate.) Note that this makes no provision for verifying that $F(x)$ is false. For the composite number example, a certificate $c(x)$ for x could be a factor of x. To verify that $F(x)$ is true, all we need to do is check that $x/c(x)$ is an integer between 1 and x.

A certificate-checking algorithm is NP if it is polynomial time and $|c(x)|$ is bounded by a polynomial in $|x|$. "NP" stands for "nondeterministic polynomial." It should be clear how *polynomial* applies to the definition, but where does *nondeterministic* come in? An algorithm that makes lucky guesses could do the hard part—that is, create $c(x)$ in polynomial time by guessing. Guessing is a nondeterministic process. Combining this with the certificate-checking algorithm gives a nondeterministic polynomial time algorithm for $F(x)$. A recognition problem is in the class NP if there exists an NP algorithm for it.

Suprisingly, there exists a class of "hardest" recognition problems in NP. These are the NP-complete problems. In what sense are they hardest? Suppose we have a g time algorithm for a problem. We say another problem is no harder than this if it has a $g(p)$ time algorithm for some polynomial p. This says that, to within a polynomial adjustment, all NP-complete problems have the same running time bound and no NP recognition problem has a larger bound. Let's put this another way. Suppose that $\mathcal{P}$ is a recognition problem that has a polynomial time certificate checking algorithm and let g be the running time for the best possible noncertificate algorithm for some NP-complete problem. Then the best noncertificate algorithm for $\mathcal{P}$ is at most $g(p)$ time for some polynomial p.

Since a polynomial time algorithm can check if $F(x)$ is true in polynomial time even without a certificate, any problem in the the class P is contained in the class NP. It's not known if the two classes are equal; however, this seems very unlikely. Why? NP-complete problems have been studied extensively and no polynomial time

algorithm has been found. (On the other hand, it hasn't been proven that a polynomial time algorithm cannot exist.)

A problem is NP-hard if it is at least as hard as an NP-complete problem. An NP-hard problem need not be a recognition problem.

Goals, Difficulties, and Compromises

Computation problems often force compromises—we saw some possible ones earlier. In fact, compromise is a pervasive aspect of AI. Being aware of this will help your understanding and creativity, so develop the habit of asking the following questions:

What are the goals?
What are the difficulties?
What are the compromises?

Try going back to the previous section and picking out the goal(s), problem(s), and compromise(s) involved in my writing of this text.

Exercises

1.4.A. Why are algorithms particularly troublesome in AI?

1.4.B. What are some ways of dealing with the problems to which algorithms in AI often lead?

1.4.C. Roughly speaking, what are NP-complete and NP-hard problems?

1.4.D. Why may it not be too important that a problem is NP-hard?

1.4.1. Prove that the anytime algorithm for finding a solution to $f(x) = 0$ has the following three properties. Assume that there is no roundoff error in the computations.

(i) There is always such an $x \in [a, b]$.

(ii) After n iterations, the length of the interval $[a, b]$ is 2^{-n} times its original length.

(iii) No matter how close to a solution of $f(x) = 0$ we want to be, we can get that close if we allow the algorithm to run long enough.

1.5 Expert Systems

"You really are an automaton—a calculating machine," I cried. "There is something positively inhuman in you at times."

—Arthur Conan Doyle (Watson to Holmes) (1889)

Expert system research has, by general consensus, not been as successful as its most vehement proponents still claim and it is open to us to wonder just why. My view is that it is due to the divergence between formalised rule and the social nature of being an expert.

—Philip Leith (1990)

Definition 1.1 Expert System

As a rough working definition, an *expert system* for some special field is an artificial system that

- exhibits abilities in that field,
- accepts input regarding a specific problem,
- delivers advice, actions, or something similar as its output, not just organized data, and
- uses domain-specific knowledge.

The traditional AI definition was more restrictive. It required that the expert system obtain its results by a process akin to abstract reasoning, that it be able to explain how it reached its conclusions, and that it exhibit abilities at least comparable to those of a human being. The abstract reasoning requirement probably arose from a combination of the desire for explanations and an intellectual prejudice concerning how AI should be done. The desire for explanations was based on the observation that people often insisted on checking the computer's "reasoning" before accepting its conclusions. Finally, if the system was not at least as good as a human in its area of expertise, no one would use it.

The broader definition given here allows for expert systems that are not based on a symbolic manipulation of data, for example, neural nets. It also allows for systems in areas where humans exhibit little if any conscious reasoning, for example, in processing visual input. Finally, it allows for useful systems that are less capable than humans but are still valuable

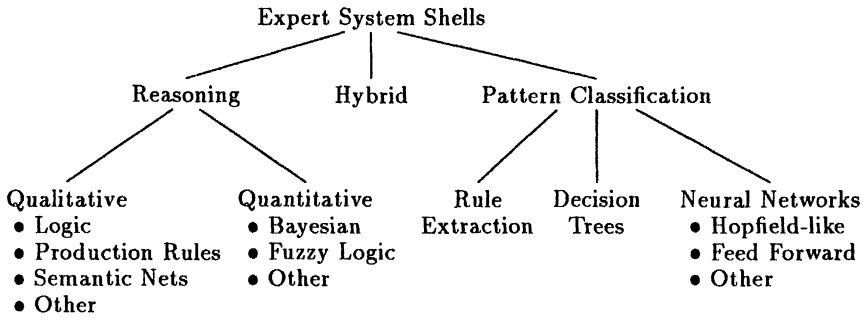

Figure 1.2 Possible engines for expert system shells. "Other" signifies the most blatant omissions. "Rule Extraction" develops input for "Reasoning" systems. "Hybrid" refers to systems that use more than one method, a practice that is becoming more common.

---

in research or applications, for example, natural language processing systems.

Closely related to the notion of an expert system is that of an *expert system shell*, which is important in the development of commercial systems. Roughly speaking,

An expert system shell is to an expert system

as

a compiler or interpreter is to a program.

In this analogy, program statements correspond to domain-specific knowledge, which is often expressed declaratively either in rules or in examples. Just like interpreters and compilers, expert system shells tend to fall into two categories:

- Rule-based knowledge is normally used at run time in a symbolic reasoning process.
- Example-based knowledge is normally used at compile time in a pattern classification process.

Figure 1.2 illustrates some of the possibilities for expert system shells.

Constructing an Expert System

There are various steps to constructing an expert system. One possible breakdown is

- selection of a tractable problem,
- selection of an appropriate shell,
- acquisition and preparation of knowledge, and
- testing.

Actually blending and feedback take place among the steps. For example, we might defer the shell choice until we have acquired some knowledge, or, in the process of testing, we might decide that the knowledge base is inadequate.

Step 1. The Problem: To begin with, you must have a "good" problem, that is, one for which an expert system is likely to be useful. How can you tell if this is the case?

Best performance is usually obtained by choosing a narrow subject; that is, one in which the knowledge base is well delimited. In particular, we should *avoid problems that involve "common sense."* Some AI problems, like natural language understanding, are plagued by the need for common sense.

Performance has generally been disappointing in areas where evolution has apparently led to some "hard wiring" in human brains. The foremost example is expert systems related to vision. Successful expert systems in such areas have generally been limited to *very* specific problems such as identifying handwritten Zip codes.

Step 2. Shell Selection: Once you've clearly stated the problem and gained some understanding of the field, you should choose a method of implementation. You can program a system from scratch, but using an approriate shell is usually much more efficient.

Step 3. Knowledge Acquisition: It is generally difficult to obtain accurate information from experts—they misstate the rules they use, forget important factors, contradict themselves (and each other), and estimate numerical values poorly. Furthermore, their knowledge is probably poorly organized for use in an expert system shell. Although the art of obtaining information from experts is important in building an expert system, we won't study it.

In some areas, such as vision processing, experts do not use conscious methods. In this case, you must either attempt to discover rules yourself or you must abandon rules and create the expert system from a collection of "typical" examples.

Step 4. Testing: Testing is often referred to as validation. You can expect problems that will send you back to Step 3 repeatedly. It is hard to decide

when a system has finally passed the testing phase and is ready for use. In the first place, users typically come up with situations that software designers did not anticipate. Second, we often do not expect 100% success, so it is hard to judge the failures we observe. In this case, systems that provide explanations are quite helpful—the reasons given for a wrong answer can help us decide if we want to attribute it to a design error or to a limitation that we cannot (or do not want to) overcome.

Examples of Expert Systems

To give you a more concrete appreciation, I'll briefly discuss a few of the many expert systems that have been written. My choices were motivated by a desire for breadth not by the commercial success, if any, of the system. Inevitably, the brevity of the descriptions has led to some distortion.

LOGIC THEORIST (1956)

In the early years of the twentieth century considerable effort was devoted to providing a solid foundation for mathematics. The most massive attempt was Whitehead and Russell's *Principia Mathematica* (1910).

Aside. This search for a solid foundation is one of the modern impossibility problems. Its impossibility was proved by Gödel in 1931. He showed that any system based on the usual methods of logical reasoning and arithmetic must contain true theorems that could not be proved within the system ("incompleteness"). In another paper, he showed that the real numbers could not be completely specified in such a system ("independence of the continuum hypothesis"). The classical impossible problems are the trisection of the angle, the doubling of the cube, and the squaring of the circle. The Renaissance impossible problem is the solving of the general fifth-degree equation by radicals. The proof that π is transcendental established the impossibility of squaring the circle. Galois theory was used to establish the impossibility of the other three problems.

Since the framework provided by Whitehead and Russell allows theorems to be proved with no "understanding" of the concepts, it's a reasonable candidate for symbolic manipulation by computer. Newell, Simon, and Shaw took up this task and produced LOGIC THEORIST.

As you've undoubtedly discovered, it's not always clear what steps must be taken to prove a theorem. Because of this, LOGIC THEORIST used an ad hoc trial-and-error method. An essential part of a trial-and-error method is deciding what to try. The program used two approaches:

- Suppose the goal is to prove Z and we have an axiom or theorem that says, "If A is true, then Z is true." We can attempt to prove A.

- Suppose the goal is to prove that "If A is true, then Z is true." We can try to find M such that one of the two statements

  "If A is true, then M is true." and "If M is true, then Z is true."

  is either an axiom or a theorem. Then we try to prove the other of the two statements.

This method is related to the reasoning engine used in the Prolog language. Unlike LOGIC THEORIST, Prolog has a theoretical foundation that provides power and clarifies its limits. On the other hand, the construction and arrangement of Prolog statements are more critical to its success.

Few, if any, researchers claim that the formal methods of LOGIC THEORIST and Prolog are used in day-to-day human reasoning. Nevertheless, a large number of researchers believe that extensions of these ideas will prove adequate for much of the reasoning needed in AI.

Mathematical logic and Prolog are discussed in Chapters 3, 4, and 6.

MYCIN (1972)

Beginning in 1972 at Stanford, Shortliffe and others developed MYCIN, which is one of the best known expert systems. In its area of competence, MYCIN was able to diagnose illnesses as well as or better than most physicians. It was also able to explain how it reached its conclusions. Nevertheless, it never received more than token acceptance from the medical community.

MYCIN is a *rule-based system* with *uncertainty*. In real life, many rules are not certain. An example of such a rule is "If you do not study, then you will get a bad grade." However, you might happen to be lucky, so the rule may be valid only 95% of the time. A typical MYCIN rule has the form

> If the result of test A is R_A and ... and the result of test Z is R_Z, then there is evidence that the disease organism is D.

Included with the rule is a numerical value in the interval $[-1, +1]$, called a *certainty factor* (CF). This value is intended as a measure of the strength of the rule's conclusion, given that its hypotheses are satisfied. In particular

$$CF = \begin{cases} +1, & \text{given the evidence, } D \text{ is certainly correct;} \\ -1, & \text{given the evidence, } D \text{ is certainly wrong;} \\ 0, & \text{the evidence gives no information about } D. \end{cases}$$

The meaning of intermediate values is not so clear.

The MYCIN reasoning engine proceeds from diagnostic evidence toward causes, eventually producing certainty factors for various diagnoses. In the process, certainty factors are combined using an ad hoc rule. More recently, certainty factors have been given a probabilistic interpretation, and Bayesian nets have provided less ad hoc (but more complex) methods for combining certainty factors.

Although numerical methods like that used in MYCIN provide ways of incorporating uncertainty into reasoning, it is unlikely that human reasoning

is based on such processes—people are notoriously poor at assigning numerical values to evidence. On the other hand, the use of numerical methods might lead to AI systems that reason more accurately than humans do.

Bayesian nets and certainty factors are discussed in Chapter 8.

NETtalk (1986)

DECtalk is a complicated rule-based system for converting written English to spoken English. Sejnowski and Rosenberg developed the *neural network* NETtalk to do the same thing. In contrast to DECtalk, NETtalk contains no rules. Instead, it contains about 100 interconnected units (neurons). The network was given paired samples of written and spoken English from which it trained itself, using a process that adjusts the strengths of the connections between the neurons. NETtalk uses the seven most recent text symbols (letters, punctuation, and spaces) to drive a digital speech synthesizer.

Networks have trained themselves for a variety of tasks. In contrast to the more cognitive approaches used in the other examples, networks have no cognitive information built in. Researchers believe that neural networks mimic somewhat the low-level behavior of biological networks of neurons. As a result, they believe that this approach may hold the key to designing AI systems that have some of the capabilities of biological systems.

Neural nets are discussed in Chapters 11 and 13.

DEEP THOUGHT (1990)

Game-playing programs were a favorite research area in the early years of AI. For various reasons, research interests have since moved in other directions, but the area has not been completely abandoned.

Chess is the most actively researched game. Programs are available that will easily beat average players. Thanks to faster processors, special-purpose devices, and improvements in programs, the top silicon-based players are now nearly as good as the top human players.

DEEP THOUGHT, by Hsu, Anantharaman, Browne, Campbell, and Nowatzyk uses special-purpose hardware to search the possibilities for several moves into the future. The quality of each possible position is evaluated and, sometimes, further search is carried out. DEEP THOUGHT's strength lies in the depth to which it can search. In contrast, Nitsche's MEPHISTO searches less and spends more time assessing the positional aspects of the situation. DEEP THOUGHT plays at or near the grandmaster level and MEPHISTO plays at a slightly lower level.

Search plays a major role in AI, but brute-force search has very limited application owing to combinatorial explosion. Some researchers believe that combining search techniques with "heuristic evaluations" and "methods of abstraction" will prove important in some parts of AI.

Search is discussed in Chapter 2 and briefly in Chapter 13.

CHATKB (1992)

Hekmatpour and Elkan developed CHATKB is an expert system to aid users of certain VLSI design tools. The rapid acceptance of this system is in marked contrast to that of others such as MYCIN. The difference may be due to the fact that CHATKB users are already using computers on a regular basis for other high-level activities such as CAD.

When faced with a user problem, CHATKB determines the category to which it belongs. This is done by an iterative questioning process similar to the game of Twenty Questions. Such processes are called *decision trees.* Nonautomated decision trees have been used for many years in natural history field guides for classifying plants and animals.

Each category contains a data base of previously analyzed problems. CHATKB finds the closest matching problem in the data base for the current problem's category. It then presents that problem and its solution to the user. If the user rejects this solution, CHATKB presents the second best match, and so on. Matching in this manner is a form of *case-based reasoning.*

Some researchers believe that this type of dichotomous approach—classify then look for similar cases—is typical of higher level day-to-day human reasoning. Consequently, they expect some such method to play an important role in the design of intelligent systems.

Decision trees are discussed in Chapter 13. Case-based reasoning is mentioned very briefly in Chapter 14.

Exercises

1.5.A. What is an expert system?

1.5.B. What are the steps in building an expert system?

Notes

Crevier [10] has written an informative, lively, nontechnical book on the history of AI based on his own background and on extensive interviews with major researchers. He brings the participants to life and accurately explains important concepts and achievements in layman's terms. You would probably enjoy it.

If my brief treatment in Section 1.1 left you dissatisfied, you may wish to look at other AI texts such as those by Charniak and McDermott [7], Firebaugh [15], Ginsberg [18], Rich and Knight [39], Russell and Norvig [41], and Winston [53].

For a discussion of topics that I've slighted, or for a less mathematical discussion of those I've covered, consult some of the available textbooks and surveys. The texts by Dean, Allen, and Aloimonds [11], Charniak and McDermott [7], Firebaugh [15], Ginsberg [18], Rich and Knight [39], Russell and Norvig [41], and Winston [53] are all broad-based introductions to AI, but the discussions of neural networks may be limited. Of these, I particularly recommend Ginsberg's and Russell and Norvig's texts. More mathematical, but less broad, are those by Dougherty and Giardina [13], Laurière [27], and Shinghal [45].

Survey and expository articles can sometimes be found in journals and in conference proceedings. Some journals, such as *Artificial Intelligence: An International Journal*, publish special issues containing several such articles. In addition, handbooks and surveys such as [3] and [46] and books such as [38] appear from time to time. For briefer discussions, there is the encyclopedia [44].

Reading original sources in any field is often a good idea, but it can be daunting because the authors usually assume readers are researchers with the necessary background. A solution is provided by annotated collections, for example, Morgan Kaufmann Publishers' *Readings in* ... books, such as [9]. A source in neurocomputing is [2].

I mentioned functional programming and logic programming as two of the paradigms that AI brought to computer science at large. Logic programming as implemented in Prolog will be discussed in Chapters 3 and 4 to help your understanding of logic. Functional programming ideas were implemented by McCarthy in Lisp. The theoretical foundation is provided by the *lambda calculus*, developed primarily by the logicians Church and Kleene. I won't be discussing these topics, but some AI texts have a Lisp-based introduction to the subject, which is good if your focus is understanding Lisp. On the other hand, MacLennan [28] discusses the general methodology of functional programming from both a concrete and an abstract viewpoint without relying on Lisp. For texts on Lisp and Prolog, see the notes at the end of Chapter 3.

The discussion about what constitutes AI continues. Almost any textbook will begin with a discussion of what AI is about and articles appear from time to time in journals and magazines; see, for example, [43]. The nature of AI and other topics of debate appear in the essays edited by Graubard [19]. These were written for a general audience. The essays in Partridge and Wilks [36] and in volume 47 of *Artificial Intelligence* (nos. 1–3, Jan. 1991) are more technical. Material on the connectionist versus symbolic debate can be found in [37] and in [40].

AI frequently employs complex nonlinear feedback systems. Their behavior is often counterintuitive—at least until extensive experimentation leads to the development of a new intuition. For the simplest such systems, mathematical control theory has produced some theoretical results. Forrester has explored complex systems by simulating corporations, cities [16], and the entire

world. Many other people have simulated complex systems and attempted to obtain heuristic principles and theoretical results. Progress has been painfully slow. It is quite possible that this area will remain intractable, but giving up now would be extremely premature.

Turing's proof of the impossibility of the halting problem depends on the concept of finite automata. You can find a proof in Bender and Williamson's text [4, pp. 178–179], any book on automata theory, or some texts on discrete mathematics. The ideas relating to Figure 1.1 are discussed more thoroughly by Russell and Wefald [42, Ch. 1]. For further discussion of NP-completeness, see the texts by Papadimitriou and Steiglitz [35] and Wilf [52] or see the article [20]. Garey and Johnson's [17] classic book on the subject lists many NP-complete problems, but the list is now *much* longer.

A discussion of expert systems can be found in many AI texts. There are also books devoted to expert systems. These include Stefik's [48] extensive introduction; the text by Jackson [24], which covers a large part of the material found in a standard AI course; the text by Lucas and van der Gaag [26], which treats fewer topics but in greater depth; and the collection [50], which discusses a variety of applications, devoting a few pages to each. The chapter discussion passed quickly over the difficult problem of knowledge acquisition. Many techniques, problems, and specific examples are discussed in [25].

Biographical Sketches

John McCarthy (1927–)

Born in Boston, he received a bachelor's degree from Caltech and a doctorate from Princeton, both in mathematics. He received the 1971 Turing Award.

McCarthy named the field; he invented the name "artificial intelligence" when writing the proposal for the first AI conference. In 1957, he and Minsky founded the Artificial Intelligence Group at MIT. While there, McCarthy invented timesharing and Lisp. In 1963, McCarthy moved permanently to Stanford, where he founded and directed SAIL (Stanford Artificial Intelligence Laboratory). The MIT and Stanford groups have had a profound influence on AI for many years.

His major concern has been understanding "commonsense" reasoning so that it can be used in AI. As a result, he's focused on achieving a fundamental understanding of knowledge and has advocated a publicly accessible knowledge base for common sense.

Many interesting stories about McCarthy can be found in the biography by Hilts [21, pp. 197–287]. The sketch by Israel [23] provides more technical information.

Marvin Minsky (1927–)

Born in New York City, he studied at Harvard and Princeton, receiving a doctorate in mathematics. As a postdoc at Harvard, he designed the first confocal microscope, a device which is now quite important in optical microscopy. Minsky received the 1969 Turing Award.

In 1957, McCarthy and Minsky founded MIT's Artificial Intelligence Group, where he has continued to inspire excellent thesis research in a variety of areas including

- MACSYMA (the forerunner of Maple and Mathematica),
- analogical reasoning (A is to B as C is to which of the following?),
- language comprehension, and
- robot vision.

Minsky introduced the idea of "frames," which are used in AI and, more recently, in object-oriented programming languages. In 1969, Minsky and Papert dealt a blow to perceptrons—a type of neural network—by proving that they were quite limited [30].

Recalling his student days, Minsky remarked that "The problem of intelligence seemed hopelessly profound. I can't remember considering anything else worth doing." [5, p. 77]. His career has focused on learning what computers are capable of doing on nonarithmetic problems.

Bernstein's [5, pp. 9–128] biographical account contains extensive quotations from Minsky.

Allen Newell (1927–1992)

Born in San Francisco, he received a bachelor's degree in physics at Stanford and began a doctorate in (pure) mathematics at Princeton. Concerned about a lack of breadth, he left Princeton for RAND where he met Herbert Simon. Newell received a doctorate in industrial administration under Simon at Carnegie Tech (now Carnegie-Mellon University), where he became a professor. Newell and Simon received the 1975 Turing Award.

He and Simon began a long and fruitful cooperation in 1955 when, with J. C. Shaw, they designed the list-processing language IPL and used it to write the LOGIC THEORIST, a program that was able to prove results found in Russell and Whitehead's *Principia Mathematica*. As a result, Newell, Shaw, and Simon are often called the parents of AI. The realization that computers are more than just rapid arithmetic calculators—that they can be used to manipulate symbols—was an important observation at the time and is now taken for granted.

In 1956, Newell, Simon, Chomsky, McCarthy, Minsky, and others launched cognitive science at a conference at MIT.

The focus of Newell's career has been the formalization of problem solving and complex task performance by human beings. The scope of this undertaking has grown over the years, moving from attempts to model the performance

in specific cognitive areas to a drive to model the entire cognitive process. This has culminated in SOAR, a blend of AI and cognitive psychology. Theories of how humans solve problems provide the motivation for this ongoing programming project whose aim is to simulate significant aspects of human cognition.

More information about Newell and SOAR can be found in [29] and about his interaction with Simon in [47].

Herbert A. Simon (1916–)

Born in Milwaukee, he studied at the University of Chicago, where he received a doctorate in political science. In his autobiography [47, p. 85], he relates that by this time he "had made a modest beginning in mathematics, a basis for subsequent self-instruction." Most of his career has been spent at Carnegie-Mellon University (CMU). Newell and Simon received the 1975 Turing Award. Simon received the 1978 Nobel Prize in Economics.

After being involved in the establishment of the CMU Graduate School of Industrial Administration, he began his shift to AI and cognitive psychology in 1955. He contributed to the establishment of CMU's fruitful interdepartmental computer science program.

As the preceding biographic sketch mentioned, Simon and Newell worked jointed for many years. But, unlike Newell, Simon has continued to focus on more limited problem-solving simulations rather than on the entire cognitive process.

Much of Simon's career has focused on the implications of "bounded rationality" in economics and cognitive science. Traditional economics postulates a very knowing and rational man; he has complete knowledge of all relevant factors, including the details of his own preferences, and is able to carry out any amount of reasoning (and computation). In the early 1950s, Simon broke with this tradition and postulated *bounded rationality*—incomplete knowledge of factors and preferences and limited reasoning abilities.

Simon's autobiography [47] is part of the Alfred P. Sloan Foundation Series—a growing collection of generally excellent autobiographies by prominent contemporary scientists.

Alan M. Turing (1912–1954)

Born in London, he took his degrees in mathematics at Cambridge, where he remained until joining the British war effort in 1938 as their first cryptanalyst. There he played a major part in setting up the system for routinely decoding the German Enigma code. After World War II, Turing spent time at the National Physical Laboratory and at Manchester.

The ACM's Turing Award is named after him, as are Turing machines and the Turing test of Exercise 1.2.1 (p. 10). (The Turing Award lectures through 1985 are collected in [1].) Turing machines illustrate Turing's focus on logic and computation. A Turing machine is an elegantly simple abstract computer. Using these simple computers, he showed that the halting problem for computer programs has no computable solution. This was done in 1935, before

the birth of the electronic computer. Using the lambda calculus, Church also showed the existence of well defined noncomputable functions. This nonexistence result has implications for first-order logic, which is the subject of Chapters 3 and 4.

Hodges [22] has published a thorough nontechnical biography of Turing. For more information on Turing machines, consult a text on automata theory.

References

1. *ACM Turing Award Lectures. The First Twenty Years: 1966 to 1985*, ACM Press, New York (1987).
2. J. A. Anderson and E. Rosenfeld (eds.), *Neurocomputing. Foundations of Research*, MIT Press, Cambridge, MA (1988).
3. A. Barr, P. R. Cohen, and E. A. Feigenbaum (eds.), *The Handbook of Artificial Intelligence*, Vols.1–4, Morgan Kaufmann, San Mateo, CA, and Addison-Wesley, Reading, MA (1981, 1982, 1989).
4. E. A. Bender and S. G. Williamson, *Foundations of Applied Combinatorics*, Addison-Wesley, Reading, MA (1991).
5. J. Bernstein, *Science Observed*, Basic Books, New York (1982).
6. K. H. Borgwardt, *The Simplex Method. A Probabilistic Analysis*, Springer-Verlag, Berlin (1987).
7. E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, MA (1985).
8. P. M. Churchland, *Matter and Consciousness*, rev. ed., MIT Press, Cambridge, MA (1988).
9. A. Collins and E. E. Smith (eds.), *Readings in Cognitive Science*, Morgan Kaufmann, San Mateo, CA (1988).
10. D. Crevier, *AI: The Tumultuous History of the Search for Artificial Intelligence*, BasicBooks, New York (1993).
11. T. Dean, J. Allen, and J. Aloimonds, *Artificial Intelligence Theory and Practice*, Benjamin/Cummings, Redwood City, CA (1994). Includes discussions of time and space complexity of AI algorithms.
12. D. C. Dennett, *Consciousness Explained*, Little, Brown, Boston (1991).
13. E. R. Dougherty and C. R. Giardina, *Mathematical Methods for Artificial Intelligence and Autonomous Systems*, Prentice Hall, Englewood Cliffs, NJ (1988).
14. M. W. Eysenck and M. T. Keane, *Cognitive Psychology: A Student's Handbook*, Lawrence Erlbaum Associates, Hillsdale, NJ (1990).

15. M. W. Firebaugh, *Artificial Intelligence: A Knowledge-Based Approach*, Boyd and Fraser, Boston (1988).

16. J. W. Forrester, *Urban Dynamics*, MIT Press, Cambridge, MA (1969).

17. M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York (1979).

18. M. Ginsberg, *Essentials of Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA (1993).

19. S. R. Graubard (ed.), *The Artificial Intelligence Debate. False Starts, Real Foundations*, MIT Press, Cambridge, MA (1988). Reprinted from *Daedalus* **117** (1988).

20. J. Hartmanis, Overview of computational complexity theory, *Proceedings of the Symposia in Applied Mathematics* **38** (1989) 1–17.

21. P. J. Hilts, *Scientific Temperaments. Three Lives in Contemporary Science*, Simon and Schuster, New York (1982).

22. A. Hodges, *Alan Turing: The Enigma*, Simon and Schuster, New York (1983).

23. D. J. Israel, A short sketch of the life and career of John McCarthy. In V. Lifschitz (ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, Academic Press, Boston (1991).

24. P. Jackson, *Introduction to Expert Systems*, 2d ed., Addison-Wesley, Reading, MA (1990). This is considerably expanded from the first edition.

25. A. L. Kidd (ed.), *Knowledge Acquisition for Expert Systems*, Plenum Press, New York (1987).

26. P. Lucas and L. van der Gaag, *Principles of Expert Systems*, Addison-Wesley, Reading, MA (1991).

27. J.-L. Laurière, *Problem Solving and Artificial Intelligence*, Prentice Hall, Englewood Cliffs, NJ (1990). Translated from the 1987 French edition by J. Howlett.

28. B. J. MacLennan, *Functional Programming. Practice and Theory*, Addison-Wesley, Reading, MA (1990).

29. J. A. Michon and A. Akyürek (eds.), *SOAR: A Cognitive Architecture in Perspective*, Kluwer, Dordrecht (1992)

30. M. L. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA (1969). It has been reprinted with some additional discussion as *Perceptrons: An Introduction to Computational Geometry, Expanded Edition*, MIT Press, Cambridge, MA (1988).

31. T. C. Moody, *Philosophy and Artificial Intelligence*, Prentice Hall, Englewood Cliffs, NJ (1993).

32. A. Newell, Unified theories of cognition and the role of Soar. In [29], 25–79.

33. A. Newell and H. A. Simon, Computer science as empirical inquiry: Symbols and search, *Communications of the ACM* **19** (1976) 113–126. This is their 1975 ACM Turing Award Lecture.
34. A. Newell, *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA (1990).
35. C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ (1988).
36. D. Partridge and Y. Wilks (eds.), *The Foundations of Artificial Intelligence. A Sourcebook*, Cambridge University Press, Cambridge, Great Britain (1990).
37. S. Pinker and J. Mehler (eds.), *Connections and Symbols*, MIT Press, Cambridge, MA (1988). Reprinted from *Cognition: International Journal of the Cognitive Sciences* **28** (1988).
38. Z. W. Ras and M. Zemankova, *Intelligent Systems. State of the Art and Future Directions*, Ellis Horwood, New York (1990).
39. E. Rich and K. Knight, *Artificial Intelligence*, 2d ed., McGraw-Hill, New York (1991). This is a major revision of Rich's first edition.
40. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing. Explorations in the Microstructure of Cognition. Volume 1: Foundations*, MIT Press, Cambridge, MA (1986).
41. S. Russell and P. Norvig, *Artificial Intelligence. A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ (1994).
42. S. Russell and E. Wefald, *Do the Right Thing: Studies in Limited Rationality*, MIT Press, Cambridge, MA (1991).
43. R. C. Schank, Where's the AI? *AI Magazine* **12** (1991) 38–49.
44. S. C. Shapiro (editor-in-chief), *Encyclopedia of Artificial Intelligence*, 2d ed., John Wiley and Sons, New York (1992).
45. R. Shinghal, *Formal Concepts in Artificial Intelligence*, Chapman and Hall, London (1992).
46. H. E. Shrobe and the American Association for Artificial Intelligence (eds.), *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA (1988)
47. H. A. Simon, *Models of My Life*, Basic Books (1991).
48. M. Stefik, *Introduction to Knowledge Systems*, Morgan Kaufmann, San Mateo, CA (1995).
49. M. A. Stillings, N. H. Feinstein, J. L. Garfield, E. L. Rissland, D. A. Rosenbaum, S. E. Weisler, and L. Baker-Ward, *Cognitive Science: An Introduction*, MIT Press, Cambridge, MA (1987).
50. E. Turban and P. R. Watkins (eds.), *Applied Expert Systems*, North-Holland, Amsterdam (1988).

51. A. M. Turing, Computing machinery and intelligence, *Mind* **59** (1950). Reprinted in [9 pp. 6–19].

52. H. S. Wilf, *Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ (1986).

53. P. H. Winston, *Artificial Intelligence*, 3d ed., Addison-Wesley, Reading, MA (1992). This is the first edition containing material on neural nets.