CHAPTER 1

INTRODUCTION: WORKING AROUND THE LIMITATIONS OF AI

At the dawn of the new millennium, there was much to marvel about in the world of computing and technology. PCs, cell phones, and other digital devices were everywhere and commonplace, enabling an unprecedented amount of digital processing and communication. The Internet had morphed, in a few short years, from a communications medium known only to a chosen few, mostly in academia and government, to a global repository of information exchange known the world over. Global positioning systems (GPS) provided everyday car drivers turn-by-turn instructions with an amazing degree of precision and accuracy. Digitization proceeded at a feverish pace, and everything from music, to videos, to the world's books were transmitted as digitized bits over miles and miles of networks, increasingly in a wireless fashion. Some would say a technology revolution had taken place, resulting in an explosion of innovative applications and ideas in the technology marketplace, unlike anything we had seen before.

Unfortunately, the trajectory of progress in artificial intelligence (AI) would be much less dramatic, and many would argue, ultimately disappointing. Amid the wild successes of the Internet and wireless communications, we would hear far less talk about machines that could think and solve problems like humans. For the most part, in recent years, AI has taken a back seat to Internet and wireless applications. Indeed, after a speculative boom in the 1980s, a time in which many of its far-out ideas did not pan out, and starting in the 1990s AI lost much of its sex appeal and dazzle. Its image and reputation as a field of promise have not recovered to this day. (It is interesting that the new overhyped application

of today is nanotechnology, another field with outsized expectations that is also likely to generate disappointing results out of alignment with public perceptions.)

It was not always so. There was once a time when AI was viewed as the wave of the future, the field that would generate the most transformative computer systems known to humankind. Perhaps no figure in the history of AI represents this promise better than Alan Turing, who in 1950 wrote the article titled "Computing Machinery and Intelligence"[1] in which he considered the question, Can machines think? In this paper, Turing formulated his famous Turing Test, a test in which a human interrogator poses questions to both a computer program and a real person. The computer passes the test if the interrogator cannot tell whether the responses come from the computer or the person. Turing was optimistic that we would have such thinking machines in the near future:

> The original question "Can Machines Think?" I believe to be too meaningless to deserve discussion. Nevertheless, I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.

As of 2009, we do not yet have machines that pass the Turing Test, let alone machines that can think.

The huge chasm between public perception and actual accomplishment in AI has only been exacerbated by science fiction and Hollywood. Many science fiction writers, such as Jules Verne and, more recently, Isaac Asimov, have long written about human-like thinking robots and have contributed to the fantasy, the hope, and to some the fear of thinking machines. Perhaps the most iconic image of our time belongs to HAL, the human-like supercomputer in Stanley Kubrick's 1968 movie *2001: A Space Odyssey*. HAL is not so much a robotic machine with a human-like physical form but a machine with a super-human red eye. It is capable of carrying on a natural conversation with other humans but can also play chess and notably possesses superior visual recognition capabilities. In one of the film's most memorable sequences, HAL reads the quick-moving lips of two of the crew members carrying on a conversation out of earshot.

The science fiction image of HAL can be contrasted to the real-life image of Grace (which stands for Graduated Robot Attending Conference), an actual robot that was created by researchers and was asked to perform at the 18th annual conference for the American Association for Artificial Intelligence held in 2002.[2] Grace took the combined efforts of five educational and research institutions to create. See Figure 1.1 for a photograph of Grace. The challenge presented to Grace was to start at the entrance to the conference center, take the elevator to the registration desk, register for the conference, and then report to the auditorium at a set time to deliver a speech.

How did Grace perform? Despite the thunderous applause that she received at the conference, an observer familiar with the hopes and dreams of AI would have been greatly disappointed. Throughout, she made a lot of mistakes. About 30 feet into her assigned task, Grace began to misinterpret the spoken commands she was getting. She bumped into walls, repeatedly stopped, or did nothing. In large part,

Figure 1.1. Grace the robot. (Reprinted with permission from Endnote 2.)

her poor performance was due to her limited voice-recognition capabilities. But more fundamentally, Grace lacked that elusive and extremely hard-to-program quality known as autonomy. Autonomy is the ability of a machine (or program) to act on its own without constant supervision. "Take the elevator to the second floor" is a command that most of us could easily execute without much effort and without constant supervision. For a robotic machine to execute such a command on its own would be a monumental and (at this point in time) impossibly difficult task.

The image of HAL vs. that of Grace is a stark one and provides us with a reality check. We are nowhere close to creating thinking machines, even by the limited standards proposed by the Turing Test. If Grace represents the state of the art, then we have a long way to go before we have designed a machine that even remotely resembles a thinking machine—one that can converse with humans, one that can recognize objects and act on them accordingly, one that possesses the motor skills required to navigate through an unknown terrain, or one that is capable of general problem-solving skills requiring only the commonsense knowledge of a child.

Why is it so hard to create a machine that can think like a human? This is a decades-old question that has plagued AI researchers from the very beginning of the field. Part of the problem lies in understanding what is meant by artificial intelligence. Definitions such as "the art of creating machines that perform functions that require intelligence when performed by people"[3] raise more questions than they provide answers about what the field is about. Perhaps a better definition is provided by Bellman: "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning."[4] But again this definition too is couched in ambiguity: What exactly do we mean by *human thinking*?

Another problem is that the definition seems to be a perpetually moving target. There is an old adage in AI that a problem is an AI problem if it hasn't been solved yet. Once some AI function is successfully programmed, it is no longer considered an AI program anymore. Unfortunately, such an attitude means that we will always be disappointed and disillusioned because "real" AI topics will be unattainable as researchers chase ever-more unpractical ideas about thinking machines and human-like robots—the AI community's version of chasing after the Holy Grail. In the meantime, real and concrete accomplishments—in areas such as expert systems, speech/voice recognition, Bayesian networks, neural networks, and other applications—will go underappreciated and even scorned by AI researchers because they will be tainted as easily solvable problems that couldn't possibly be of real interest to AI.

Such thinking is misguided and damaging to the field of AI, both to its perception as a viable research area, as well as an area full of promising applications. This book argues that we need to have a clear-sighted understanding of the current limitations of AI and the difficulties of creating thinking machines. Rather than bemoaning these limitations and simply giving up, or pursuing unrealistic AI agendas to the exclusion of all others, we need to design AI systems, in particular the user interface, to take into account these limitations and difficulties. Indeed, we need to be more embracing of the shortcomings by finding workaround ways to make AI systems more useful and usable given these inherent shortcomings.

THE DIFFICULTIES OF CREATING A THINKING MACHINE

One common view that explains the difficulty of creating a thinking machine is that AI programs lack commonsense knowledge. Commonsense knowledge, as opposed to specialized expert knowledge, refers to the things that most people can do, often without conscious thought.[5] In fact, when you ask people to explain their commonsense reasoning, they are often at a loss to do so because such knowledge has become so automatized that they lose conscious access it.

As an example, interpreting English sentences such as the following will happen automatically to an average reader of English:

Last night Mary and Jane went out to dinner. It was Mary's birthday, so Jane paid for dinner. When the check came, she offered to pick up the tab.

A reader of the above three sentences is likely to infer the following:

1. Yesterday was Mary's birthday.
2. It is customary, according to culture and tradition, for the person having a birthday to be treated for dinner.
3. *She* in the third sentence refers to Jane, not Mary.
4. A *tab* is the same thing as a check. To *pick up the tab* refers to paying the bill.
5. The check came at the end of the dinner, not at the beginning.

When one considers all the knowledge that is required to interpret English sentences, it is no wonder that we have been unable to create a reliable language translation program. The irony is that even though a child early on learns to use and understand thousands of words, no computer is capable of understanding what those words mean and cannot even carry on a simple conversation. (The Turing Test, it turns out, has been a tough nut to crack!) To create a language translation program, say from French to English, a programmer might start by creating a simple dictionary look-up program, mapping French words to English words, and rearranging the words according to the laws of English grammar and usage. In fact, that is just what early efforts at language translation programs were. However, such a program is bound to generate a translation that is rife with errors and awkward constructions. (The famous mistranslation of "*the spirit is willing, but the flesh is weak*" into "the vodka is good, but the meat is rotten" illustrates this point well.) One obvious problem is that the program does not contain knowledge of all the idiomatic expressions of the English language. But a bigger problem is that the program lacks a mental model* of the world being discussed. A language translation program, unlike a human, makes no use of models of the world to understand language and will thus easily become overwhelmed by ambiguities and multiple meanings. Indeed, early efforts at creating language translation programs failed miserably and disappointed AI researchers.[6]

What kinds of models might a language translation program contain? For one, it might contain a knowledge structure known as a semantic network, a kind of diagrammatic representation that is composed of nodes and links that interconnect the nodes. Many kinds of semantic networks are possible. See Figure 1.2 for one type of semantic network. In one simple form, the nodes are represented by rectangles and can be anything from a physical object (such as a book, house, tree) to a person (such as Abraham Lincoln, Mary, a waiter), to a concept (such as happiness, religion, crime) or to an event (such as yesterday, Fourth of July, the first day of school). The links interconnect two or more nodes and show the relationship between the nodes (sometimes a verb or other piece of descriptive text is added to describe the relationship). Figure 1.2 represents a simple semantic network of the "Mary and Jane went out to dinner" scenario described earlier.
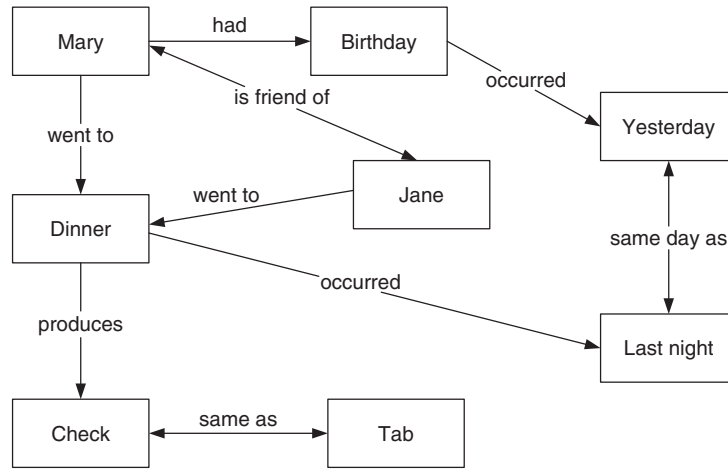
*Mental models are discussed in Chapter 2.
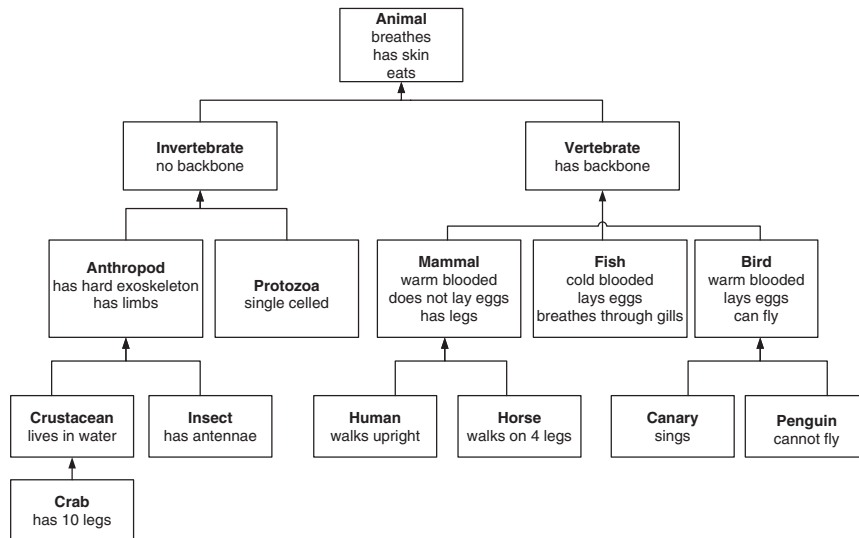
Figure 1.2. A semantic network.

Figure 1.3. An inheritance hierarchy.

A second type of of model is an inheritance hierarchy that can represent a taxonomy or classification scheme. See Figure 1.3 for an example of this type of mode. In this hierarchy three pieces of information are represented:

- **Units**. The thing or object.
- **Properties**. Characteristics of the unit.
- **Pointers**. Class associations among the units.

In this example, knowledge about animals can be illustrated as a hierarchy of nodes, represented as rectangles in the diagram. A node connected to another node is either on the top (a superclass or parent node) or on the bottom (a subclass or a child node). For example, the superclass *animal* is connected to two subclasses, *invertebrate* and *vertebrate*. Under this scheme, subclasses inherit all properties of their superclasses. (Properties are indicated below the class name in each of the nodes in the figure.) For example, while *canary* inherits properties from *bird, vertebrate*, and *animal*, it does not inherit properties from *fish*. Note also that a particular unit on the hierarchy can override a property it inherits: *Penguin* inherits the property *can fly* from its superclass *bird*, but it is overridden by the property *cannot fly*.

A third type of model that might be included in a language translation program might be a script, a well-known knowledge structure described by Schank and Abelson.[7] A script can be defined as a stereotyped sequence of events that together define a well-known situation. The best-known example is the one for going to a restaurant. A restaurant script might include the following sequence of events:

1. Enter the restaurant.
2. Be greeted by the host, who seats you at a table and gives you a menu.
3. Read menu and decide what you want.
4. Order when the waiter comes.
5. Wait for food to come.
6. Eat when food comes.
7. Ask for the check.
8. Pay the check.
9. Exit the restaurant.

The advantage of having a set of scripts is that they can give an AI program the ability to infer actions and predict what will happen in conventional situations.

It is a rather trivial matter to construct semantic networks, inheritance hierarchies, and scripts for very simple situations and problem solving contexts. However, to construct them for all of a typical human adult's knowledge about the world would be a colossal undertaking. Consider for example that if a typical adult knows thousands of words, each of those words would be linked to hundreds of knowledge structures, which in turn are linked to hundreds of other knowledge structures, and so on, resulting in a very intricate and elaborate network. Moreover, these semantic networks are not static but typically change over time as a person acquires experience and more knowledge about the world. A more practical use of semantic networks, inheritance hierarchies, scripts, and other knowledge structures would be to construct them for more well-defined contexts and problem solving tasks.

Lack of Commonsense Reasoning

Several AI researchers have tackled the problem of understanding commonsense reasoning. Among some of the components that are frequently brought forth as essential components of commonsense reasoning are the following:[8]

- The ability to use different representations to describe the same situation.
- The ability to recognize when a problem solving method is failing.
- Self-reflection at a higher level.
- Efficient knowledge retrieval.

Let us look at each of the four issues and consider what is wrong with the current generation of AI systems. Further, let us suggest some possible remedies for these limitations. These four components of commonsense reasoning serve as a framework for how to evaluate the effectiveness of an intelligent user interface.

The Ability to Use Multiple Representations. A conventional program is built such that it approaches a problem in a single way. Such a program is likely to be rigid and not very robust to changing assumptions and to different ways of using the program. We need AI programs that are built to work around multiple representations, so that when one representation fails, the program is easily able to switch over to find an alternative. For a human problem solver to really comprehend a problem-solving situation, he or she must possess not only a set of representations on the problem but also an ability to understand how they are related to one another, a kind of integrated view of the set. Likewise, an AI program should be able to understand how to process multiple representations effectively.

The topic of knowledge representation is a central one in AI, one that has occupied the attention of AI researchers from the very beginning, and one that remains a central challenge today. One goal of this book is to address how to approach the topic of knowledge representation as refracted through the prism of diagrammatic representations. In the chapters to follow, we look at many different types of knowledge representation schemes, all of them represented diagrammatically. We investigate how to reason with and draw inferences on many different kinds of graphical diagrams. Some examples include decision trees and influence diagrams that help us make better decisions (Chapter 3), directed graphs to aid us in understanding mechanisms of cause and effect (Chapter 3), Venn diagrams that can be used to perform logic reasoning (Chapter 4), diagrams to represent the problem-solving strategies used in expert system (Chapter 6), model-based reasoning systems that can help us with fault diagnosis (see Chapter 7), and Bayesian networks that perform probabilistic reasoning (or reasoning with inexact and imperfect data) (Chapter 8). Others have looked at neural networks, scripts (briefly discussed earlier), frame-based representations, logic, object-oriented approaches, and other techniques and formalisms for representing knowledge. Because the focus of this book is on reasoning with diagrams, many of these additional knowledge representation techniques will not be covered insofar as they do not deal with diagrammatic representations.

The Ability to Deal with Errors. Traditional AI user interfaces have long been criticized for being extremely brittle—that is, they are programmed to do only one specific thing, but when you try to push the system beyond what it was programmed to do (i.e., beyond its bounds of ignorance), the system will completely malfunction. Indeed, the AI landscape is littered with thousands of highly specialized programs that can each do some well-circumscribed task: diagnosing bacterial skin infections, playing chess, determining what drug to take and in what dosage for patients with high cholesterol. Such programs can perform their specific task very well, sometimes even surpassing the level of a human expert. However, if even one underlying assumption is unmet, if one piece of data is missing, or if some piece of evidence is not known with complete certainty, the system will completely break down. Human problem solvers, by contrast, degrade more gracefully when faced with uncertainty, incompleteness in data, or noisy information. Moreover, human problem solvers are better equipped to know when failure occurs and can employ adaptive strategies to deal with such situations.

We need to do better job when designing our AI systems. In great part, the ability to deal with error and failure can be vastly improved with a better user interface, today this being primarily the graphical user interface. In general, we need user interfaces that are flexible enough to process errors and uncertain information. We deal with the subject of system flexibility at the end of this chapter and throughout this book.

Self-Reflection at a Higher Level. A conventional program executes a pre-programmed set of instructions but no attempt is ever made by the program to stop and reflect on its own actions, even when it is pursuing a blind alley or veering wildly off track. Such a program is badly in need of some kind of over-all plan of attack, a model of how to approach a problem-solving situation. A human problem solver often possesses higher-level strategies on how to approach a problem. He or she will often need to stop and reflect on the current state of the problem and decide whether a change of course is in order. Having strategic knowledge and the ability to adapt in midstream are important characteristics of effective problem solvers and decision makers.

We need AI programs that possess strategic knowledge and can reflect on whether they are pursuing the right problem-solving methods. Strategic knowledge is about understanding the methods used for problem solving and about how those methods are ordered to reach a goal.[9] Strategic knowledge and how to diagrammatically represent such knowledge are addressed more extensively in Chapter 6.

Efficient Knowledge Retrieval. The ability to retrieve relevant information from a person's vast storehouse of knowledge is, of course, an important function for effective performance and problem solving. AI programs also need to do to be able to do this effectively, so that it is relatively easy to get at the right information quickly (an exhaustive and linear search through the memory, or

knowledge base, of an AI program will just not cut it). Moreover, knowledge retrieval requires that AI programs can recognize patterns and features of a current problem-solving context so that the program knows which problem context stored in its knowledge base will best match the current context.

A number of knowledge representation schemes have been proposed to deal with the knowledge retrieval problem. One popular method is known as case-based reasoning. This type of system organizes knowledge into cases, which are problem solving experiences used for inferring solutions to future problems. An example of an application is a technical support system that provides help desk operators guidance on how to troubleshoot a customer's problems. For instance, a customer calls a help desk to report a problem with her printer. She reports that she is receiving error message 908, that her printer does nothing, and that the orange light is flashing. A case-based reasoning system will search through its database of historical cases to find the one that best matches the features of the current problem. At times, there will not be a perfect match to an historical case, so the system must know how to adapt the previous solution to fit the current problem. In addition, a new case that does not perfectly match any of the previous cases is added to the knowledge base of historical cases, so that a case-based reasoning system adapts and grows over time as it acquires more and more experiential knowledge.

This book investigates a variety of diagrammatic representations, one of whose primary functions will be to serve as "organizational scaffolds"[10] that enable more efficient knowledge retrieval. Indeed, a very important function of diagrams is to organize information more effectively so that it is easier to retrieve. For example, one common and natural way to organize a lot of information is as a hierarchy. These diagrams, whether hierarchy or some other form, can be part of the user interface itself, enabling an end user to interact with the diagram to better understand the system. Hence in addition to their role as efficient knowledge retrievers, they will also serve in the role of helping us understand the components of a system and their interrelationships—that is, the underlying mechanism of a complex system.

Intractability

Another explanation for the difficulty of AI and the difficulty of creating thinking machines is the intractability of many of its problems. These are problems that are computationally difficult, if not impossible to solve, with currently known algorithms. Whereas the problems of AI programs lacking commonsense reasoning concentrate on general problem-solving skills that almost everyone possesses (even a child), intractable problems can involve more complex and specialized problem-solving tasks (such as playing chess).

Early on in the history of AI, most research efforts were centered on very simple, toy problems, also known as microworlds, so that computational complexity was not a primary concern. A good example is the **blocks world**, a domain that consisted of a set of blocks (e.g., rectangular blocks, cones, balls, etc.) placed
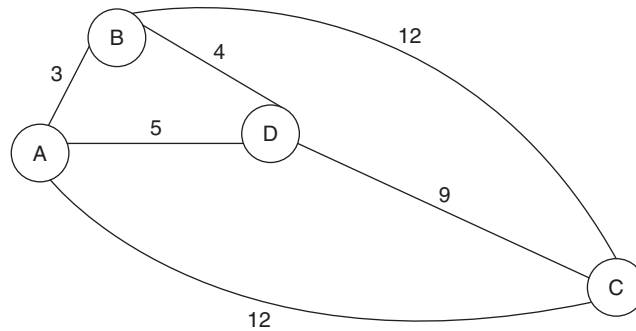
Figure 1.4. Traveling salesman problem. What is the solution to this problem given four cities, A, B, C, and D? Answer: One solution is ACDBA (28 miles).

on a tabletop. A typical task in this domain was to command a robotic arm to rearrange the blocks in a certain way. For example, "place the cone on top of the large block." SHRDLU was a natural language program that allowed user interaction with the blocks world.[11] The user instructed SHRDLU to manipulate the various objects in the blocks world. The program was computationally feasible because the blocks world was so simple: The entire set of objects could be described by perhaps 50 words, from nouns like *block* and *cone* to adjectives like *small* and *red*. Unfortunately, researchers at the time failed to realize that such simple toy problems would not scale up to more realistic and complex problem-solving domains without becoming computationally intractable.

Intractability has had a great impact on the field of AI and computer science. Many problems, to the consternation of many early AI prognosticators, have turned out to be too difficult to solve, a turn of events that has greatly limited the progress of AI. One classic example that illustrates the nature of intractability is the well-known traveling salesman problem. In this problem, we are given n cities, and for each pair of cities, the distance between the two cities. The problem is to find the shortest round-trip route that visits each city once and returns to the starting city. An example is given in Figure 1.4.

One way to solve this problem is simply by brute force search—that is, try out every possible ordering of the cities and select the one with the minimum total number of miles. This approach will work fine only when the problem size is small but quickly becomes intractable. For $n = 4$, you would need to try out 4! permutations or 24 possibilities.* In general, you would need to try out n! permutations, for a problem size of n cities. The problem quickly grows in size as n increases. For example, for $n = 10$ cities you would need to try out 3.6 million permutations and for $n = 20$ cities, the time complexity of the problem grows to

*In the case of 4 cities, there are 4 possible cities to be chosen as the first city. Once this city has been selected, there are $(n − 1)$ or 3 cities to choose from the remaining cities. Once this city has been removed from the list, there are now $(n − 2)$ or 2 cities. Hence, in general, for a problem of size n there are n! or $n(n − 1)(n − 2) \cdots 1$ permutations.

2.4×10^{18} permutations. Obviously, a brute force search strategy would quickly bog down even the fastest computers of today for sufficiently large values of n.

How does one define intractability? One definition is that a problem is intractable if the time required to solve the problem grows exponentially with the size of the problem.[12] By exponential time, mathematically we mean that computation time $T(n)$ is a function of problem size n and there exists a constant $c > 1$ such that:

$$T(n) = O(c^n)$$

Polynomial time functions, by contrast, are much more desirable in terms of time complexity. Mathematically they can be written as:

$$T(n) = O(n^k)$$

Examples of exponential time functions include 2^n and 3^n. Examples of polynomial time functions include n (linear time), n^2 (quadratic time), and n^3 (cubic time).

A side-by-side comparison of polynomial time functions versus exponential time function is given in Table 1.1.[14] In the table, the functions express execution times in terms of microseconds. The interesting thing to observe here is the extremely rapid growth rates of the exponential time functions, compared to the polynomial time functions. It is quickly evident that polynomial time functions are much more desirable than exponential time functions. As Gary and Johnson state, "Most exponential time algorithms are merely variations on exhaustive search, whereas polynomial time algorithms generally are made possible only

Table 1.1. Comparison of several polynomial and exponential time complexity functions. Adapted from Endnote 13

Time complexity function	Size n					
	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00005 second	.00005 second	.00006 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

through the gain of some deeper insight into the structure of a problem. There is wide agreement that a problem has not been 'well-solved' until a polynomial time algorithm is known for it."[14] (It is worthwhile to note that using dynamic programming, a mathematical method for solving problems with a sequential decision structure, the traveling salesman problem can be solved in time $O(n^2 * 2^n)$. Although still exponential, it is better than $O(n!)$.)

It is beyond the scope of this book to identify and address the problem of intractability. The theory of NP-completeness,[15] which is not addressed in this book, was developed for this purpose and is discussed in many standard texts covering AI algorithms. We do not deal with issues of intractability any further in this book. Certainly, diagrammatic approaches may be used to better understand the structure of a problem and suggest ways to develop heuristics, or rules of thumb, to make some of these problems more manageable, by suggesting good-enough solutions, but not necessarily optimal solutions. In Chapter 7, we employ model-based reasoning, a technique that starts with a diagrammatic model of some kind and reasons from this model to help us solve an intractable problem. Hence diagrammatic methods may help us better understand intractable problems as well as provide insight as to how to find work-around solutions.

EXPLANATORY POWER OF INTELLIGENT SYSTEMS

It is obvious that there are huge difficulties that need to be overcome before AI programs resemble thinking machines. Many of these obstacles will remain insurmountable for the foreseeable future at least, despite all the best efforts of AI researchers and the business community to create more robust AI programs. Breakthroughs in areas such as computer vision, language understanding, and machine learning as well as better AI algorithms to deal with intractability will all contribute to the progress of AI, and hopefully narrow the gap that exists between the clunky programs and robots of today and the AI programs of tomorrow. It is not the goal of this book to address what needs to be done to bridge this gap and create machines and programs that can more closely resemble human thinking. Rather, this book turns its attention to what will increasingly become an important and central component of AI systems: **the user interface**, in particular the graphical user interface, which will increasingly use diagrams and various other graphical representations to aid in system understanding.

As AI technologies become more widely used in the future, the graphical user interface is likely to take on a more prominent role. There are at least two reasons for this prediction. First, given the limitations of AI systems—and the unlikelihood of their being resolved in the near future—there will be a more urgent need to create user interfaces that are better able to cope with an AI system's weak spots. As discussed earlier, AI systems that are rigid dialogues are unable to deal gracefully with their lack of commonsense reasoning—such as, their inability to process uncertain and incomplete data and their inability to retrieve knowledge efficiently. A user interface that can, at least in part, deal with these weaknesses

is far more likely to be effective and accepted by the user community. Second, the AI systems of the future are likely to tackle increasingly complex tasks, capable of solving problems in domains as diverse as medical diagnosis, computer design/configuration, and loan portfolio analysis, just to name a few examples. If users are to trust and accept the advice and recommendations that these systems generate, the user interface will need to be able to comfortably explain itself.

What do we mean by a system that can explain itself? It is useful at this point to limit our discussion to advice-giving systems that provide recommendations on how to solve problems or help us make the right decisions. Hereafter, such systems are referred to as **intelligent systems**. Such intelligent systems are said to be endowed with **explanatory power** when they are capable of explaining their own actions through a user interface.[16] Two system characteristics are relevant to understanding explanatory power: **transparency** or the ability to see the underlying mechanism of the system so that it is not a black box and **flexibility** or the ability of the user interface to adapt to a wide variety of end-user interactions so that it is not a rigid dialogue but an open-ended interaction that allows the user to explore and understand the system more fully. While system transparency is a quality related to the informational content of the system itself—that is, information that helps us understand a system's actions—flexibility is more related to the nature of the end-user interaction with the system. More flexibility in the user interface can lead to more transparency—having a more open-ended interaction can enable a user to seek out more ways to better understand the system. By the same token, having a more restrictive interface can impair a user's ability to seek out more transparency, even it does already exist. Hence flexibility is treated as a separate quality of the user interface that exists independently of interface transparency.

System Transparency

How can we render an intelligent system more visible for inspection so that its internal mechanism is no longer a black box? System transparency involves providing users with information about how a system functions. In general, as typical users of computers and technology, we are often plagued by a computer's lack of transparency. We usually aren't bothered by this state of affairs until the system malfunctions or when we wish to use a system in a novel or nonroutine way. If this is the case, we may yearn for some kind of deeper system understanding that will help us cope with the error or novelty. Most times, we receive no information at all regarding what a system is doing. If we are fortunate enough to get any message at all, it will oftentimes make no sense to us, or will not contribute to a deeper understanding of the system. For example, suppose an unusual error message pops up on your computer screen: "Buffer overflow. Please restart computer." What does the message mean? What is a buffer? What was the cause of the error? And what does restarting the computer do to resolve the problem? Unless there is system transparency, we are relegated to the role of passive user, unable to do anything but blindly accept the system's recommended action.

To provide system transparency, a number of questions could conceivably be asked by a user and answered by an intelligent system:

- How did you reach that conclusion?
- Why did <such and such> an event occur?
- What if I tried <such and such>?
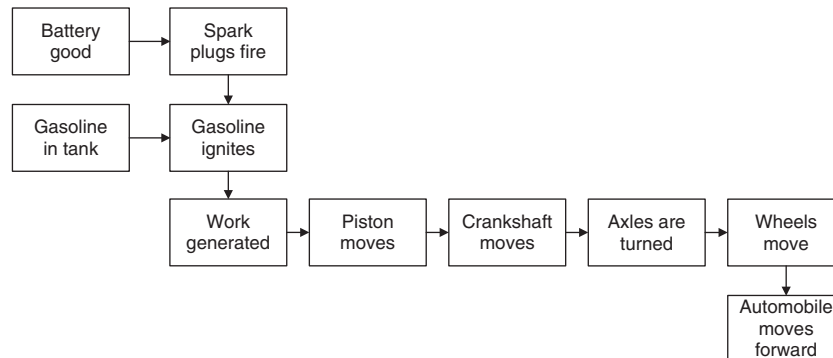- Why not <such and such>?

It would be a very difficult task indeed to devise a system that could answer all these questions in a satisfactory manner, even for a relatively simple domain. Questions of the last type (*Why not?*), in particular, are notoriously difficult to answer because they require an anticipation of alternative hypotheses. One approach would be to create a database of questions and their corresponding answers, a kind of FAQs (frequently asked questions) for the situation at hand. A problem with this approach is that a system designer would need to anticipate all questions and their responses beforehand. Yet it is highly unlikely that all the relevant questions will be thought out ahead of time. Furthermore, the responses to the questions would be canned text, or prefabricated responses, which would ignore context and contain no model of system behavior. This approach, while certainly better than having no explanations at all, is limited in effectiveness and cannot scale up to more complex domains or anticipate future scenarios, particularly novel situations that have not been encountered before.

A second possibility, one that may provide more flexibility than an enumeration of questions and answers, and one that can provide a more expansive and global view of a system, is to provide the end-user with a conceptual model of the domain. A good conceptual model can provide one with a deeper understanding of a system. The conceptual model may, for example, contain a description of the components of a system, how they are interconnected to one another and how they may causally interact to produce outputs. A conceptual model is typically rendered as a graphical diagram of some kind. One simple example is given in Figure 1.5, which portrays the causal chain of events that causes a car to move. It is based on the following description of how an internal combustion engine moves an automobile forward:

> In an internal combustion engine, gasoline is ignited by a spark plug, causing a small explosion, which generates work. The work pushes a piston, which in turn drives a crankshaft connected to an axle. The axle is connected to wheels, which turn to drive the automobile forward.

We will address the topic of conceptual models and causal models more fully in Chapter 2. Then, in Chapter 3, we will look at a wide range of diagrams, and attempt to provide a classification of their different forms, by how they are used, and what they are suitable for. There are many possible ways for promoting deep understanding with a diagram of some kind.

```
┌──────────┐      ┌──────────┐
│ Battery  │─────▶│  Spark   │
│  good    │      │plugs fire│
└──────────┘      └──────────┘
                        │
┌──────────┐      ┌──────────┐
│ Gasoline │─────▶│ Gasoline │
│ in tank  │      │ ignites  │
└──────────┘      └──────────┘
                        │
                  ┌──────────┐   ┌──────────┐   ┌──────────┐   ┌──────────┐   ┌──────────┐
                  │   Work   │──▶│  Piston  │──▶│Crankshaft│──▶│ Axles are│──▶│  Wheels  │
                  │generated │   │  moves   │   │  moves   │   │  turned  │   │  move    │
                  └──────────┘   └──────────┘   └──────────┘   └──────────┘   └──────────┘
                                                                                    │
                                                                              ┌──────────┐
                                                                              │Automobile│
                                                                              │  moves   │
                                                                              │ forward  │
                                                                              └──────────┘
```

Figure 1.5. Causal model of how a car moves.

For now, we note that a good conceptual model can facilitate system understanding—for example, an understanding of mechanisms of cause and effect. By possessing such an understanding, a user is better equipped to deal with system errors and problems. Moreover, a good conceptual model can predict the effects of our actions. Without a conceptual model, a user must operate by rote, blindly accepting a system's recommendations.

System Flexibility

The careful selection of features of the user interface may also enhance its explanatory power. A highly flexible system may encourage exploration and experimentation, which means that users will more likely seek out explanations and deeper meanings in the system recommendations. A rigid interface, by contrast, discourages them from seeking out more information because the effort to do so would be too great and not worth it. Moreover, highly flexible systems can adapt to different problem-solving contexts and to different informational needs. Let us look at, specifically, six desirable user interface characteristics[17] that may lead to greater system flexibility: (1) natural mappings, (2) feedback, (3) recoverability, (4) granularity, (5) ability to handle errors, and (6) multiple representations to the same knowledge.

Natural Mappings.   The term *natural mappings* refers to the ability to represent the user's domain so that it maps as closely as possible to his or her mental model. In his book *The Design of Everyday Things*, Donald Norman[18] provides a number of examples of everyday objects that are easy and natural to use. One reason that these objects are so easy to figure out is that they possess a natural mapping that is visible and intuitive enough to understand. For example, when we look at a knob, we assume that it is for turning and adjusting something (e.g., the flame on a stove). Likewise we assume a slot is for inserting things, a switch turns something on or off, and a handle on a cabinet door is used to pull it open. In

these examples, the physical appearance of the object, by itself, naturally suggests to us what its purpose is, and how it is to be used.

We should expect the same for intelligent user interfaces. For example, a text-based, question-and-answer dialogue may be the primary vehicle for entering inputs and displaying outputs. Such an interface would work well enough if all we wanted to do was enter inputs and have an intelligent system display a recommendation. However, the rigidity of the question-and-answer format may not be a natural way to foster explanation-seeking behaviors. For more flexibility, a graphical user interface could allow users to perform actions directly on a graphical diagram of the domain. Such an interface could facilitate a more naturalistic interaction with the system. Throughout this book, we will look at a number of diagrammatic user interfaces and consider how they may be used to support a more naturalistic dialogue with the user.

Feedback. The ability to send back information about what action has been performed and what has been accomplished is *feedback*. It is a well-established principle in user interface design, and for good reason: Without feedback, a user may wonder whether anything has happened yet. Feedback is missing in all too many systems. For example, we often experience the frustration of receiving no feedback when we click on a button in a web page to perform some kind of action (such as downloading software) and nothing happens. Sometimes we end up repeating the mouse-click, and the operation, to our dismay, is performed twice. Or we refresh the web page, thinking erroneously that the web page is stuck.

Feedback is critical in intelligent systems. It may give us comfort that we are on the right track; it may guide us toward system understanding; it may help in building system trust. For maximum effectiveness, feedback should be provided immediately, and whenever there is a change in system state.

Recoverability. Recoverability is the ability to back out of changes made to the system. This capability may encourage a user to explore and experiment with a system, without having to start all over again when an error is made. There are many possible ways to build recoverability into an interface. As users of today's most popular software products today, we are familiar with many of them: a back button, undo facilities, history lists of the most recent actions performed, hypertext links that enable a user to jump directly to a relevant portion of a problem-solving situation.*

Another approach to promoting exploratory behaviors is to create special modes of operation. For example, a protected mode might be created in which user actions are shielded from harmful consequences. A user could turn on protected mode and system commands are simulated without actually affecting the

---

*A good example of software that has these features is TurboTax. The software allows you to jump to a specific portion of your tax return—such us your deductions—so that you do not have to search sequentially through every portion of your tax return if you are interested in seeing and modifying only one portion of it.

data.[19] Another possibility is control blocking. This means that a portion, or subset, of a system's functions is made inaccessible to the user to prevent their accidental usage. These and other techniques can be used to encourage system exploration: Users are freer to actively experiment with a system's features, without fear of destroying data, or producing irreparable damage to the system.

Granularity. The ability to display different levels of detail depending on the situation the user is currently in is granularity. Especially in complex systems composed of multiple components, this capability is very useful in addressing the cognitive limitations of a user. Such a user can become overwhelmed by the enormous amount of information required to understand the system. An interface that supports granularity may be one that allows the creation of multileveled, hierarchic descriptions of a system. A user can drill down the branches of the hierarchy to obtain more detail, or go up the branches to obtain a higher-level view. An interface endowed with granularity would enable a user to change the level of detail frequently and with ease during a user session.

Ability to Handle Errors. As discussed previously, traditional AI programs are unable to handle errors well. In such programs, the user is required to enter a set of inputs in a prescribed sequence, and there may be no tolerance for uncertainty in the data, or missing data. A flexible system might possess the following desirable characteristics:

- We should be allowed to enter data in any order, and the system recommendation should not be affected by the order (invariance to data entry order).
- We should be able to enter incomplete data, and the system should make its best guess as to what the system recommendation should be given what it does know (ability to process incomplete information).
- We should be able to attach certainty factors* to our data inputs if we are uncertain about the accuracy of the data (e.g., we may have an imprecise test result, or we may have a data sensor that is not 100% accurate); the system will accordingly adapt its recommendation (ability to process noisy data).
- We should be able to perform sensitivity analysis on the data, to test how robust a system recommendation is to a range of values (ability to test for the robustness of the system recommendation).

Beyond these specific measures and recommendations, it is important for designers of intelligent systems to design for error. Having such a design philosophy

---

*A certainty factor expresses how certain we are about the data inputs. There are different methods used to assign certainty factors and process them. For now, we may think of a certainty factor as being in the range from 0 (completely uncertain) to 1.0 (completely certain). We will have more to say about certainty factors in Chapter 8.

will predispose designers to the idea that people will misuse intelligent systems, misconstrue its features, and sometimes get lost when using these systems. Designers aware of these human tendencies can plan for them ahead of time and alleviate their adverse consequences. This could make all the difference in terms of whether these systems are ultimately embraced or not by the user community.

Better still, it is highly recommended to perform extensive end-user testing before systems are implemented and deployed to the larger user population. One goal of such testing would be to come up with a list of errors that users commonly make. Having knowledge of the most common types of errors could be used to redesign the user interface so that committing errors does not result in complete system breakdown. This important step, regrettably, is all too often neglected and ignored in the development of intelligent systems.

Multiple Representations to the Same Knowledge. Because different tasks and different users may have different requirements for using knowledge, a system that enables multiple representations to the same knowledge would permit greater flexibility. We have already identified the ability to use multiple representations as a component of commonsense reasoning and have noted that this book will address many types of diagrammatic representations. We will see throughout this book how these graphical representations can be employed to support different kinds of users and different kinds of tasks.

THE FUTURE OF AI: TOWARD INTERACTIVE DIAGRAMS

We began the chapter by observing the huge discrepancy that exists between the futuristic thinking robots so vividly portrayed in science fiction and the current reality of limited AI machines and programs that can solve only well-circumscribed tasks. We further noted that this has been the cause of disillusionment and disappointment to many in the field who got caught up in the hype and excitement of robots and thinking machines. However, such an unrealistic outlook on the future of AI downplays the real and concrete achievements that have occurred in the past decades since AI was born in the 1950s. Here are just a few notable success stories:

- The chess program Deep Blue, created by IBM, defeated Gary Kasparov in 1997 (the program was capable of evaluating some 200 million board positions per second).
- Neural networks are now capable of recognizing speech, albeit in well-structured domains with limited vocabularies.
- Expert system technology has been successfully implemented in many business organizations and has resulted in higher productivity and better and more consistent decision making, not to mention that it has shed considerable light on the nature of expertise and how humans solve problems in complex domains.

- NASA sent autonomous vehicles (rovers) into space to act as robots that could perform on-site geological investigations on Mars.

All these applications illustrate a few key commonalities about what makes an AI program successful. First, they deal with well-defined domains, as opposed to tackling general problem solving. This is an important point, because all these programs focus their efforts on resolving one specific problem, not creating a machine that can do everything a human can. Second, they recognize the limitations of AI, but nonetheless manage to work around them. Their successes are highly attributable to having a well-grounded understanding of the practical issues involved in implementing AI programs and machines. A central tenet of this book is that the designers of the next generation of AI programs will require a good grasp of the difficulties of AI to find work-around solutions. In this chapter, we have discussed two frameworks for dealing with the difficulties of intelligent systems.

The first framework explores an AI program's lack of commonsense reasoning. We looked specifically at four constituents of commonsense reasoning—namely, the use of multiple representations, the ability to deal with failure, self-reflection, and knowledge retrieval. Intelligent systems in the future will increasingly need to be able to deal with these four problem areas if they are to have any chance of solving complex problems in realistic settings.

The second framework explores the concept of explanatory power, or the ability of a system to explain itself. The focus of the discussion here turned squarely on the user interface, which will increasingly assume a central role in AI systems. We looked specifically at two characteristics of the user interface: transparency and flexibility. Intelligent user interfaces of the future that are endowed with these characteristics are more likely to be embraced and accepted because users are more likely to understand and, therefore, trust their recommendations. Moreover, these systems will be able to handle a greater variety of problems and degrade more gracefully when faced with errors and uncertainty. AI systems, in general, have long been regarded as inflexible and poorly adaptable in dealing with new situations.

One solution to these limitations is to develop some kind of diagram or graphical model that serves as the central part of the user interface, the primary means by which an end-user will interact with a system. A diagram is defined as a graphic representation that shows how something works or makes something easier to understand. This book focuses not only static diagrams, such as the type that might exist in the pages of a book, but also on dynamic diagrams that can be part of an intelligent user interface. An end-user can explore and manipulate these more dynamic diagrams in a variety of ways.

In the chapters that follow we investigate a wide variety of diagrams that can be used in intelligent user interfaces. One important characteristic of the diagrammatic interfaces that we will explore is that they are meant to *serve in an advisory capacity* to human decision making and problem solving. This is an important point to underscore, because these systems are not meant to replace the human,

contrary to futuristic AI notions of robots and programs working autonomously, on their own without any kind of supervision. Indeed, the user remains firmly behind the steering wheel, in the decision making loop, throughout. The belief here is that a collaboration between human and machine is much more effective than either one working alone, and that each party brings different capabilities to the table. In effect, this means that a human user need not blindly accept an intelligent system's recommendations and conclusions, but can actively question, probe, and try to understand rationales underlying system actions. It is the interactive diagram that will make the partnership between human and machine possible.

ENDNOTES

1. Alan M. Turing, "Computing Machinery and Intelligence," *Mind* 59, no. 236 (1950), 433–460.

2. Curtis Gillespie, "Charmed by Six Feet of Circuitry," *New York Times*, Aug. 8, 2002. Photo by Rick MacWilliam. Reprinted with permission from Edmonton Journal.

3. Raymond Kurzweil, *The Age of Intelligent Machines*. Cambridge, MA: MIT Press, 1990.

4. Richard E. Bellman, *An Introduction to Artificial Intelligence: Can Computers Think?* San Francisco, CA: Boyd & Fraser, 1978.

5. Marvin Minsky, "Commonsense-Based Interfaces," *Communications of the ACM* 43, no. 8 (Aug. 2000), 67–73.

6. Douglas R. Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid*. New York: Vintage Books, 1979.

7. Roger C. Schank and Robert P. Abelson, *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: L. Erlbaum Associates, 1977.

8. See Minsky, pp. 69–70, for a discussion of some of the components of commonsense reasoning.

9. William J. Clancey, "The Epistemology of a Rule-Based Expert System—A Framework for Explanation," *Artificial Intelligence* 20 (1983), 215–251.

10. The phrase *organizational scaffold* is from Bower, 41. He remarked: "a hierarchy is an extremely familiar and efficient organizational scaffold." Gordon H. Bower, "Organizational Factors in memory," *Cognitive Psychology* 1 (1970), 18–46.

11. Terry Winograd, "Understanding Natural Language," *Cognitive Psychology* 3, no. 1 (1971), 1–191.

12. Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2003.

13. Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

14. Ibid., p. 8.

15. For a good introduction to the theory of NP-completeness see Garey and Johnson. See also S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proceedings New York: ACM of the 3rd Annual ACM Symposium on Theory of Computing*.

New York: ACM, (1971), pp. 151–158; and R. M. Karp, "Reducibility Among Combinatorial Problems, in *Complexity of Computer Computations*. New York: Plenum Press, 1972, pp. 85–103.

16. For more details on explanatory power see Robbie T. Nakatsu, "Explanatory Power of Intelligent Systems," in *Intelligent Decision-Making Support Systems: Foundations, Applications and Challenges*. London: Springer, 2006, pp. 123–143.

17. The framework for these interface characteristics is borrowed and adapted from Marilyn Stelzner and Michael D. Williams, "The Evolution of Interface Requirements for Expert Systems," in *Expert Systems: The User Interface*. Ablex Pub. Corp.: Norwood, NJ, 1988, pp. 285–305.

18. Donald A.Norman, *The Design of Everyday Things*. New York: Basic Books, 2002, pp. 1–9.

19. A. P. Jagodzinski, Theoretical Basis for the Representation of On-Line Computer Systems to Naïve Users, *International Journal of Man-Machine Studies*, 18, no. 3 (1983), 215–252.