DIAGRAMMATIC REASONING IN AI

Robbie Nakatsu

A JOHN WILEY & SONS, INC., PUBLICATION
DIAGRAMMATIC REASONING IN AI
CONTENTS

PREFACE vii

CHAPTER 1 INTRODUCTION: WORKING AROUND THE LIMITATIONS OF AI 1

CHAPTER 2 MENTAL MODELS: DIAGRAMS IN THE MIND’S EYE 23

CHAPTER 3 TYPES OF DIAGRAMS 57

CHAPTER 4 LOGIC REASONING WITH DIAGRAMS 108

CHAPTER 5 RULE-BASED EXPERT SYSTEMS 143

CHAPTER 6 RULE-BASED REASONING WITH DIAGRAMS 188

CHAPTER 7 MODEL-BASED REASONING 228

CHAPTER 8 INEXACT REASONING WITH CERTAINTY FACTORS AND BAYESIAN NETWORKS 264

CHAPTER 9 A FRAMEWORK FOR UNDERSTANDING DIAGRAMMATIC REASONING 302

INDEX 321
This book is really the end product of over a decade of work, on and off, on diagrammatic reasoning in artificial intelligence (AI). In developing this book, I drew inspiration from a variety of sources: two experimental studies, the development of two prototype systems, an extensive literature review and analysis in AI, human–computer interaction (HCI), and cognitive psychology. This work especially contributes to our understanding of how to design the graphical user interface to support the needs of the end user in decision-making and problem-solving tasks. These are important topics today because there is an urgent need to understand how end users can cope with increasingly complex information technologies and computer-based information systems. Diagrammatic representations can help in this regard. Moreover, I believe that reasoning with diagrams will become an important part of the newest generation of AI systems to be developed in the future.

I began investigating the topic of diagrammatic reasoning several years ago as a doctoral student while working on research on user interface design. Almost serendipitously, I stumbled on a concept in cognitive psychology known as mental models. This is the idea that we construct models of the world in our minds to help us in our daily interactions with the world. I was intrigued by the idea and wanted to learn more about how, when, and why people do this. I believed that if we could better understand what these mental models are about, then we might use this knowledge to design computer user interfaces and aids, such as tutorials and explanations that might support people in complex tasks, and in their everyday lives. I devote an entire chapter (Chapter 2) to the subject of mental models.

It turns out that my investigation on mental models naturally and gradually evolved into a more general investigation on diagrams. This is because I soon
came to view mental models, in many cases, as nothing more than diagrams in the mind’s eye. By diagram I mean a graphical representation of how objects in a domain are interconnected or interrelated to one another. (In Chapter 3 I try to pin down the concept of diagram by defining it more precisely. I also provide a taxonomy of diagram types.) In the process of researching and studying diagramming, I made a few key discoveries. The first is that diagramming is really a basic human activity—most of us do it quite naturally, even if in an informal and ad hoc way. On occasion, we do it more formally and explicitly and will spend time to create the “right” diagram, especially if we need to present it to others. I was surprised at how often the need to diagram appeared in my own daily life; it was not at all difficult to come up with several examples of diagramming. The second discovery is that diagramming can take on numerous incarnations and forms, more than I could ever imagine. It was overwhelming to keep track of all the different notations and techniques. Yet, underlying all these variations in notation were a few underlying principles and themes. I will address what these principles and themes are later on in Chapter 3 and in the concluding chapter, Chapter 9. The third discovery is that a diagram is more than just a static picture or representation. Diagrams can be used in more dynamic and interesting ways. Indeed, a theme of this book is that diagrams can be a central part of an intelligent user interface, meant to be manipulated and modified and, in some cases, used to infer solutions to difficult problems. All in all, there is much more to diagramming than meets the eye.

What were my motivations for writing this book and what message do I want to convey to the reader? First, I wanted to understand how diagrams can be used to help learners understand complex ideas. As a teacher at an institution of higher education, I am particularly interested in the pedagogical function of diagrams—to teach and communicate complex ideas with precision and clarity. I am keenly aware of the difficulties that students face in the classroom in trying to understand course material. Textbooks, all too often, contain explanations that confuse more than edify, and classroom lectures often fail to communicate effectively because instructors make too many assumptions about what students are supposed to know or what they already know. In the end, the classroom environment fails to create an effective mental model of the course material for the learner. Moreover, interconnections and interrelationships among concepts are not reinforced strongly enough, so that retention of the material is short lived. Throughout this book, I present many examples of how diagramming can be used to convey information more effectively to learners.

A second motivation is to understand how to make AI systems easier to understand and use. This is really one of the primary objectives of the book. Many critics of AI systems have argued for more transparency and flexibility in the user interface if users are to embrace and accept these systems. Traditional intelligent systems are black box systems that provide little or no opportunity to actively probe and question system conclusions and recommendations. Therefore, I argue that a diagrammatic user interface can help users better understand and visualize system actions.
To this end, I borrow heavily from AI and hence the title of the book is *Diagrammatic Reasoning in AI*. I could just as easily have titled the book simply “Diagrammatic Reasoning” or “The Visualization of Expertise,” but these titles do not adequately capture how much I have borrowed from the AI discipline. I look, specifically, at expert systems, model-based reasoning, and inexact reasoning (including certainty factors and Bayesian networks)—three important AI areas that have attempted to create programs capable of emulating human thinking and problem solving in various ways. I also cover logic reasoning (Chapter 4), which is a topic that has also been dealt with extensively in the AI literature.

A third motivation for writing this book is that there are no books that I know of on the marketplace today that address diagrammatic reasoning in a coherent or unified way. I hope to fill this void by providing a more cohesive treatment of the subject. While there are a number of books about information design and graphic design that deal with the topic of diagramming, they explore the topic primarily from the perspective of illustrating principles of good graphic design. There are also several books that deal with specific diagramming notations. For instance, there are books on Unified Modeling Language (UML), a diagramming standard used to model software systems and aid in systems development. Another diagramming technique that is well covered in the literature is the decision graph and other notations used for decision analysis. All these books do a fine and nimble job of describing and illustrating one specific type of diagramming technique, but they are limited because they deal with only one type of diagram or focus on one type of reasoning methodology. This book, on the other hand, is intended to cover a diverse range of diagrams and reasoning methodologies, thereby exposing the reader to the larger issues surrounding diagrams. I hope that by presenting many different types of diagrams and many types of applications, the reader will come away with a deeper appreciation of the power of diagrams.

The targeted audiences of this book are practitioners and researchers in AI and human–computer interaction, programmers and designers of graphical user interfaces (including designers of web applications), and business and computing professionals who might be interested in deploying intelligent systems in their organizations. This book is also suitable for noncomputing professionals who are interested in learning more about the power of diagrams. Indeed, unlike many AI texts on the marketplace today, I assume no prior knowledge of AI or mathematics beyond high school algebra. (The one exception is when I discuss Bayesian networks, a topic that requires a basic understanding of probability theory; in Chapter 8, I provide a brief introduction to probability theory for the reader who has no prior knowledge of the subject.) This book may be used as a self-learner’s guide to diagrammatic reasoning and intelligent user interfaces. Furthermore, the diagrammatic applications developed in this book are not targeted to any one particular audience but were created to represent diversity and to demonstrate that diagramming can be a powerful technique for everyone.

The book consists of nine chapters, each of which is more or less self-contained, so that the reader can easily read any one of them, in any order, without any knowledge of the prior chapters. One exception is Chapter 5, which
PREFACE
describes the fundamentals of rule-based expert systems; this information serves as background knowledge for Chapter 6. The other exception is the final chapter, Chapter 9, which is meant to serve as a culmination of the previous eight chapters.

Chapter 1 begins with a discussion of the difficulties of AI and the limitations of creating machines that can solve problems like humans (the so-called thinking machine that Alan Turing proposed decades ago). I argue in this chapter that we need to be more accepting of the limitations of AI by finding work-around solutions. In particular, I suggest that we need to look at the role the user interface plays in an intelligent system: How can we make intelligent systems more transparent and more flexible so that we are more accepting of their limitations?

Chapter 2 looks at mental models, or internal models, that we create in our minds to understand a complex phenomenon or system. I have subtitled this chapter “Diagrams in the Mind’s Eye” to reflect the idea that a mental model often involves the creation of an adhoc diagram, created on the fly, to help us solve problems and respond to the real world. I will look at several examples of how this occurs and why mental models are useful for problem solving. The discussion will center on two types of mental models:

- **Internal connections.** A description of how the components of a system causally interact to produce outputs or behaviors.
- **External connections.** The connections between a person’s prior knowledge and a complex target system to be understood (e.g., the use of analogical representations to understand a complex domain).

I illustrate mental models with several examples, including a mental model of an electromechanical thermostat (an original example) and several well-known examples from the cognitive science literature, including the use of analogies to help with creative problem solving.

Chapter 3 classifies the great variety of diagrams in use today. The classification scheme consists of six categories of diagrams according to their function:

- System topology.
- Sequence and flow.
- Hierarchy and classification.
- Association.
- Cause and effect.
- Logic reasoning.

Throughout this chapter, I illustrate diagrams in a wide range of application areas: everything from network diagrams that show how the hardware components in a computer network are interconnected to one another, to flowcharts that help doctors classify heart attack patients, to semantic networks that illustrate how the characters of Shakespeare’s play *Hamlet* are related. Even with such a tremendous diversity of usage, these six categories cut across all these types of diagrams and
serve as a unifying framework for understanding and organizing a great variety of diagramming notations.

Chapter 4 is about the use of diagrams in formal logic reasoning. This chapter demonstrates how complex logic problems can be made more comprehensible through the use of diagrams. I look especially at dynamic Venn diagrams that can be used to construct logic proofs for a subset of logic problems.

The discussion of Venn diagrams takes the reader, step by step, through a procedure that enables one to construct logic proofs in a graphical and visual way. The Venn diagrams described in this chapter go far beyond the conventional Venn diagram used in mathematics and set theory: They are not merely static diagrams that depict the relations between two or three sets, rather they are meant to be modified, updated, and combined in many different ways. In fact, I show how Venn diagrams can be used to construct valid logic proofs. I then consider how a linguistic representation system, such as first-order logic, compares to a nonlinguistic system, such as Venn diagrams.

Chapter 5 describes expert systems, which are AI programs that emulate the decision-making ability of a human expert, a person who has expertise in a certain area. In many respects, this entire book is about capturing expertise in some form or another, and thus expert systems play a central role in the discussion. The most common form of expert system stores knowledge as a collection of if–then rules; hence they are referred to as rule-based expert systems. In this chapter, I describe the components of a traditional rule-based expert system, including its two most important parts, the knowledge base and the inference engine. I then illustrate how to create a rule-based expert system using the specialized programming language CLIPS. Finally, I consider some of the benefits and problems of expert system technology.

Chapter 6 explores some of the techniques that may be employed to increase the transparency and flexibility of rule-based expert systems. Specifically, I look at a number of diagrams that can serve as the central component of the user interface itself, including

- Flowchart diagrams that allow one to visually trace the line of reasoning through the knowledge base, taking the user step by step through the reasoning process.
- Diagrams in which a complex knowledge base is partitioned into meaningful segments that can be organized in a hierarchic way.
- Rule trace diagrams that graphically show the interrelationships between the conditions and actions in a rule trace.
- Diagrams that model strategic knowledge, the methods and approaches used for problem solving, so that users have a high-level sense of how the expert systems is reaching its conclusions.

These diagrammatic user interfaces enable a user to more effectively visualize how a system is reaching its conclusions and recommendations. Further, these
user interfaces are highly flexible because they allow the user to explore and test out different scenarios and assumptions.

Chapter 7 discusses model-based reasoning techniques and how they may be employed to create more interactive intelligent systems. This technique offers a powerful alternative to more traditional rule-based representational systems. By model-based reasoning, I am referring to a class of AI techniques that involves the analysis of both the structure and the behavior of a system. Model-based reasoning systems start out with some kind of diagram and then reasons with the diagram to help solve difficult problems. We will look at two applications of model-based reasoning. First, we will look at how model-based reasoning can be used to aid in the fault diagnosis of a simple device. Second, we will look at how model-based reasoning can be used to help in the design of business logistics networks.

Chapter 8 delves into the problem of inexact reasoning or how to represent and process uncertainty in AI. This chapter describes two different approaches to processing uncertainty—namely, certainty factors and Bayesian networks. The first approach, certainty factors, was developed as a practical and convenient way for processing uncertainty. Although it is easy to compute certainty factors, this approach lacks rigor and theoretical justification. Therefore, Bayesian networks, an approach that has become increasingly popular today, is described as an alternative that offers a more technically correct approach. Its calculations are based on probability theory and Bayes’ theorem.

In addition, I look at how these two approaches can be modeled using belief networks and causal diagrams. These diagrams are not merely static but are dynamic because they can change based on the introduction of new data and evidence.

Finally, in Chapter 9, I summarize and integrate the discussion of the previous eight chapters. I attempt to address the following questions: What is the essence of diagramming? What are the criteria for good diagrams? How do we classify the diagrammatic reasoning techniques covered throughout the book? By answering these questions, I hope to provide a framework for understanding diagrammatic reasoning.

An important part of the book is the development of applications and graphical illustrations throughout. I draw on such diverse areas as physical science, macroeconomics, finance, business logistics management, and medicine to illustrate some of the key ideas. For example, I use diagrams and graphical illustrations to illustrate what factors affect the unemployment rate in the United States. (What are the variables and how do you graphically depict causal relationships among the variables?). In the medical domain, I illustrate a decision flowchart that predicts what factors predict a heart attack. The decision flowchart is meant to be used by emergency room personnel who must quickly make decisions about what to do with patients who come to the emergency room with symptoms of a heart attack.

Unless otherwise noted, most of the diagrams in the book are original examples. I thought that it was very important, if I was going to write a book on
diagrams, to develop original examples and applications. I also believed that it was important for me to actually draw the diagrams—only then would I be fully aware of the benefits and limitations of a particular diagramming technique. Hence all of the original diagrams were manually drawn (with the help of Microsoft Visio). In drawing the diagrams, I learned that only through the active creation of diagrams is one able to appreciate that diagramming is a process, sometimes requiring iteration and refinement. This is especially true for more complex diagrams, many of which do not fit on a single page. The end product, the diagrams that you see in the pages of this book, were sometimes arrived at through a consideration of difficult design trade-offs. I discovered very quickly and early on that no one diagramming notation is perfect or complete and that, in the end, what you see in this book is a final result of these trade-offs.

Robbie Nakatsu

February 2009
Los Angeles, California

ACKNOWLEDGMENTS

I am grateful to a number of individuals who assisted in the publication of this book. Thanks to Chris Green, who designed the book cover, and the cover department at Wiley for producing the final book cover that you see. A number of individuals read portions of the manuscript and offered their useful suggestions and comments. I am especially grateful to Peder Fedde, who read through some of the diagrams and chapters in the book to ensure that they were accurate and clear. He was extremely supportive and helpful throughout; his support was especially valuable during those times when the progress of the book seemed to move very slowly. Izak Benbasat of the University of British Columbia supported the development of the prototype systems (TransMode Hierarchy and LogNet) discussed in the book while I was a doctoral student there.

I am fortunate to be a part of a very supportive network of colleagues at Loyola Marymount University. They have been a source of support and inspiration to me through these years. I am grateful to the Summer Research Grant Committee of the College of Business for awarding me a grant to pursue the writing of this book. I would also like to thank my students, both graduate and undergraduate, for their support. Two in particular, Timothy Lui and Nathan Peranelli, served as my undergraduate research students. Glenn Grau-Johnson and Ted Tegencamp helped research some of the copyright issues related to the publication of this book. Diana Asai provided excellent administrative support. Tony Patino and his marketing class provided valuable comments on how to market and promote the book.

The staff at Wiley have been very helpful and professional throughout the process. I want to thank George Telecki, Associate Publisher, for having confidence in the book, even when it was in its initial, unformed stages. Kristen
Parrish, Senior Production Editor, was outstanding in overseeing the production of the book, and was always patient, responsive, and helpful to any concerns that I had. Lucy Hitz, Editorial Assistant, unfailingly fielded all my questions and took the time to oversee many aspects of the book. I would also like to thank Candace Levy, the copy editor, who offered many useful suggestions on how the manuscript could be improved. I feel fortunate to have had the opportunity to work with such a team of professionals.

Finally, I am most grateful to my parents, Ron and Yoshiko Nakatsu, for their support throughout the years. This book is dedicated to them.
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
INTRODUCTION: WORKING AROUND THE LIMITATIONS OF AI

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” 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. 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,
INTRODUCTION: WORKING AROUND THE LIMITATIONS OF AI

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.
INTRODUCTION: WORKING AROUND THE LIMITATIONS OF AI

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”\textsuperscript{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.”\textsuperscript{4} But again this definition too is couched in ambiguity: What exactly do we mean by \textit{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.\textsuperscript{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.*
A second type 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. 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 overall 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. 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”\textsuperscript{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. 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 \times 10^{18} \text{ permutations. Obviously, a brute force search strategy would quickly}
\text{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
\text{intractable if the time required to solve the problem grows exponentially with}
\text{the size of the problem.}^{12} \text{ By exponential time, mathematically we mean that}
\text{computation time } T(n) \text{ is a function of problem size } n \text{ and there exists a}
\text{constant } c > 1 \text{ such that:}

\[ T(n) = O(c^n) \]

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

\[ T(n) = O(n^k) \]

\text{Examples of exponential time functions include } 2^n \text{ and } 3^n. \text{ Examples of poly-
\text{nomial time functions include } n \text{ (linear time), } n^2 \text{ (quadratic time), and } n^3 \text{ (cubic}
\text{time).}

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

<table>
<thead>
<tr>
<th>Time complexity function</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>.00001</td>
<td>.00002</td>
<td>.00003</td>
<td>.00005</td>
<td>.00005</td>
<td>.00006</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>.001</td>
<td>.004</td>
<td>.009</td>
<td>.016</td>
<td>.025</td>
<td>.036</td>
</tr>
<tr>
<td>( n^3 )</td>
<td>.001</td>
<td>.008</td>
<td>.027</td>
<td>.064</td>
<td>.125</td>
<td>.216</td>
</tr>
<tr>
<td>( n^5 )</td>
<td>.001</td>
<td>1.0</td>
<td>17.9</td>
<td>12.7</td>
<td>35.7</td>
<td>366</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>.001</td>
<td>58</td>
<td>6.5</td>
<td>3855</td>
<td>2 \times 10^8</td>
<td>1.3 \times 10^{13}</td>
</tr>
<tr>
<td>( 3^n )</td>
<td>.059</td>
<td>1.5</td>
<td>6.5</td>
<td>3855</td>
<td>2 \times 10^8</td>
<td>1.3 \times 10^{13}</td>
</tr>
</tbody>
</table>
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 $O(n^2 \times 2^n)$. 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
INTRODUCTION: WORKING AROUND THE LIMITATIONS OF AI

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 if 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.