Advances in Decision-Theoretic AI: Limited Rationality and Abstract Search

by

Michael Patrick Frank

S.B., Symbolic Systems
Stanford University, 1991

Submitted to the Department of Electrical Engineering
and Computer Science in partial fulfillment
of the requirements for the degree of

Master of Science
in Computer Science
at the

Massachusetts Institute of Technology

May 1994

Signature of Author____
Department of Electrical Engineering and Computer Science
May 6, 1994

Certified by ____
Jon Doyle
Principal Research Scientist, Laboratory for Computer Science
Thesis Supervisor

Accepted by____
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Studies

Advances in Decision-Theoretic AI:
Limited Rationality and Abstract Search

by

Michael Patrick Frank

Submitted to the Department of Electrical Engineering and Computer Science on May 6, 1994 in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Abstract

This thesis reports on the current state of ongoing research by the author and others investigating the use of decision theory—its principles, methods, and philosophy—in Artificial Intelligence. Most of the research discussed in this thesis concerns decision problems arising within the context of Computer Game-Playing domains, although we discuss applications in other areas such as Clinical Decision-Making and Planning as well. We discuss in detail several AI techniques that use decision-theoretic methods, offer some initial experimental tests of their underlying hypotheses, and present work on some new decision-theoretic techniques under development by the author.

We also discuss some attempts, by the author and others, to transcend a few of the limitations of the earlier approaches, and of traditional decision theory, and synthesize broader, more powerful theories of abstract reasoning and limited rationality that can be applied to solve decision problems more effectively. These attempts have not yet achieved conclusive success. We discuss the most promising of the current avenues of research, and propose some possible new directions for future work.

Thesis Supervisor:

Dr. Jon Doyle
Principal Research Scientist, Laboratory for Computer Science

Acknowledgments

First and foremost, a very great debt of thanks is owed to my advisor, Jon Doyle, for introducing me to the field of theoretical AI, for teaching me much of what I know about it, and for serving as the guiding light for my explorations in that area over the last three years. His careful, critical thinking and his broad knowledge of the field have kept me from making many egregious errors and omissions. Any such that may remain in this thesis are solely due to my tardiness in giving it to Jon for review, and are entirely my own fault. I especially appreciate Jon's patience throughout the last year, when, at times, it must have seemed like I would never finish.

I would also like to thank the rest of the members of the MEDG group within the last three years, especially Peter Szolovits, for his dedicated and tireless leadership. Together they have provided an enormously friendly, supportive, and stimulating atmosphere for research, and have always been willing to listen to my crazy ideas: Isaac Kohane, Bill Long, Vijay Balasubramanian, Ronald Bodkin, Annette Ellis, Ira Haimowitz, Milos Hauskrecht, Scott Hofmeister, Yeona Jang, Tze-Yun Leong, Steve Lincoln, Scott Reischmann, Tessa Rowland, Whitney Winston, Jennifer Wu, and the rest. Peter, Ron, and Whitney, especially, greatly influenced my work by introducing me to many interesting ideas. Milos deserves an extra "thank-you" for bearing with me as my officemate.

I must also express my enormous gratitude to my friends Barney Pell, Mark Torrance, and Carl Witty, for being my peers and partners throughout my academic career. Carl has also been a most excellent roommate throughout my time at MIT, and has also made a number of intellectual contributions acknowledged within this thesis. I would also like to thank my pen-pals Chris Phoenix and William Doyle for lots of email correspondence that has helped to keep my fingers in practice.

Thanks also to Steve Pauker at NEMC for taking time out of his busy schedule to meet with me, and to Eric Baum and Warren Smith at NEC for giving me the chance to work with them firsthand last summer. I am also grateful to all the MIT professors I know, who have provided some of the best and most interesting classes I have ever taken.

In addition to infinite gratitude due my entire family for all they have given me throughout my life, I wish to give an extra special thanks to my mother Denise, my father

Patrick, and to Bruce, for bearing with me during some particularly difficult moments in the last year, despite going through some even more difficult troubles themselves.

Finally, most of all, I want to thank Jody Alperin, with more warmth than I can ever express. Without her I would never have survived. She has been the source of boundless joy and inspiration for me, with her irrepressible delight, her fierce protectiveness, her brave determination, and her eager intellect. Despite having herself gone through the worst things imaginable, she has given me all the best things imaginable. She has kept me going when I thought nothing could, and in finding my love for her, I have found myself. I hereby dedicate this thesis to her.

Table of Contents

Chapter 1

Introduction

One of the primary goals of artificial intelligence (AI) has always been to make machines that are capable of making good decisions. For example, an ideal AI medical diagnosis and treatment program would need to decide what possible diseases to consider as explanations for the patient's symptoms, what tests to administer to gain more information about the patient's condition, and what medical procedures to perform to alleviate the patient's condition. Or, on a somewhat less critical level, a good chess-playing program would need to decide which lines of play to consider, how much time to spend studying the different possible alternatives, and finally what move to make.

But good decisions are not easy to come by. Economists have studied human decision-making for many years in an effort to understand the economic decision-making behavior of individuals and groups. Drawing from the philosophy of utilitarianism and the mathematics of probability, they have devised an idealized theory of optimal decision-making, called decision theory (DT). But the advent of decision theory has not suddenly made human decision-making an easy thing, for several reasons. One problem is that, although decision theory can help you make decisions when your problem involves unavoidable uncertainties, such as the outcome of a dice roll, it does not directly tell you how to cope with avoidable uncertainties, where you could gain more information before making the decision. You can try to apply decision theory recursively to the problem of deciding what information to gather and how, but this leads to the second problem of decision theory, which is that thoroughly analyzing a decision problem in accordance with the theory can be an arbitrarily long and complex and time-consuming process, so that it would make more sense to just do a simpler, but less accurate analysis.

This is the conundrum of limited rationality. Our brains, and our computers, do not have unlimited speed when reasoning or computing. And neither do we have unlimited time to make our decisions. But using decision theory to its fullest extent can require an indefinitely-large amount of computation. So, paradoxically, it can be a bad decision to try to make a very good decision. We can never know that applying the theory in any particular case is a better decision than not applying it and just guessing instead. Decision theory really has nothing to say about how to resolve this issue. It does not answer the question of its own use by limited entities such as ourselves and our creations, the computers.

Perhaps partly because of this difficulty, and also because of the extreme mathematical complexity of the more advanced decision-theoretic methods, researchers in AI have not typically used decision theory extensively in the design or operation of their programs. In fact, for many years decision theory (and the foundational disciplines such as probability lying behind it) were almost completely ignored by the AI community. Instead, theoretical AI has focused on a variety of other mathematical frameworks that eschewed the use of quantitative, numeric entities such as probabilities and utilities, and dealt instead with symbolic, discrete, logical entities such as beliefs, plans, and preferences.[1] In the meantime, the more applied, empirical segments of the AI community did not use decision theory either; they pursued other methods in essentially a trial-and-error fashion, taking whatever techniques seemed to work well and refining and developing them, regardless of whether the technique had any kind of deeper theoretical basis or justification. As a result, it is really not well understood why these techniques worked (when they did), and so it was difficult to gain any deeper understanding from writing a program to solve one problem (such as playing chess) that would help in writing a program to solve a different problem (such as clinical decision making).

So, perhaps because of this difficulty of generalization and reuse of the existing techniques, in recent years AI researchers have begun to take another look at foundational theories such as decision theory, despite those theories' problems. The medical AI community has been using decision theory for some time (e.g., [Gorry-et-al-73], [Schwartz-et-al-73]. See [Pauker-Eckman-92] for a brief overview), and now AI researchers are even applying DT to computer game-playing, an area considered by many to be a core testbed for AI techniques, although it is questionable how much transference of ideas there has

1. A somewhat questionable focus, considering that the sciences seem to have always moved from the qualitative to the quantitative, and the quantitative methods, in my opinion, have yielded the lion's share of science's success and power. But perhaps it was just too early in the development of AI for quantitative theories to be fruitful.

been from the specialized realm of game playing programs to other areas of AI. Still, the successful demonstration of these general techniques in game playing could serve to excite many people about those techniques, and lead to their adoption in other domains as a foundation for further progress.

In the work reported in this thesis, I have performed preliminary investigations in several areas related to the use of decision theory in AI. First, I have studied the traditional computer game playing methods and their evolution towards the newest approaches, which make extensive use of decision theory. In this thesis I describe the evolution of the different approaches, and report on the results of new experiments I performed to assess the accuracy of the hypotheses underlying several of them. The recent use of decision theory in game-playing has met with some success, and more importantly, the ability to transfer ideas to the solution of problems other than game playing seems to be much greater with the decision-theoretic methods than with the earlier game playing techniques. However, game-playing algorithms still suffer from a lack of generality, so I have also worked on improving the generalizability of game programs by allowing them to deal with the playing problem on a more abstract level. I report here on my investigations of "abstract search" algorithms. Second, in their attempt to apply decision theory effectively, computer game-playing (CGP) researchers have worked on alleviating decision theory's limited rationality problem. In an effort to understand the generality of their solutions, I have done preliminary investigations of how to transfer the CGP limited-rationality techniques to another decision problem, that of medical decision making. I have also investigated issues concerning the formalization of limited rationality theory, and its possible development into a more practical successor to decision theory. These, too, appear to be promising lines for further study. A primary conclusion of this thesis will be that the right kind of limited rationality theory will involve abstraction in a crucial way.

Properly using the tools of probability and decision theory is not easy, and a future theory of limited rationality would likely be even harder to apply. But from what I have seen so far, these quantitative, foundational theories seem to be a good basis upon which to develop programs that successfully solve decision problems. I believe that eventually, like other sciences, artificial intelligence will make a transition to a state where its theoretical models and empirical methods are precise, stable and quantitative. For AI is really just another natural science, one that studies a phenomenon—intelligent thought—that occurs naturally, in humans, if nowhere else. Just as physicists learned about universal laws of motion, electricity, etc., through observations of very simplified physical systems in the laboratory,[1] so too, AI can be seen as attempting to discover universal laws of thought by studying simplified forms of reasoning within the "laboratory" of the computer.

1.1 Structure of This Thesis

Chapter 2, "Evolution of Computer Game Playing Techniques," begins with a discussion of the evolution of decision-theoretic methods and the concurrent development of automated reasoning algorithms, particularly for playing games. I report here for the first time some results of experiments performed by myself and a colleague on some of these algorithms. Interestingly, decision theory and game theory historically share a common genesis, but the applied, computational side of game theory then diverged from decision theory, and has only recently begun to bring decision theory back into game-playing with recent attempts to use it at the "meta" level to control the computational reasoning itself.

These recent developments resurrect the old issue of the limited nature of rationality, and the need for a successor of decision theory that could accommodate it. These matters are discussed in Chapter 3, "Limited Rationality," along with some of the preliminary attempts by the author and others to devise such a theory. Many issues remain to be resolved in this area, and they appear very difficult to surmount.

However, regardless of whether a good theory of limited rationality is attained that could form a coherent basis for game-playing programs and other reasoning programs to control their own computational reasoning, the question remains as to whether the whole traditional state-space search oriented approach to game playing and most other kinds of automated reasoning is really appropriate. Many researchers have noted that a more general and efficient method based on concepts such as patterns, "chunking," or abstraction might be possible to develop. Much theory and practical methodology in this area remains to be developed, but Chapter 4, "Abstract State-Space Search," describes some of the preliminary developments in this direction.

---

1. For example, Galileo's experiments on gravity that were conducted by rolling objects down inclined planes.

So, where is this all going? Chapter 5, “Summary and Conclusions,” sums up and presents my vision of some of the AI research of the future that will be most fruitful for myself and others to explore.

1.2 Related W ork

A more detailed overview of classical computer game-playing than the one presented in Chapter 2 can be found in [Newborn-89]. The book [Marsland-Schaeffer-90] contains a good selection of fairly recent readings, and the proceedings of the Spring 1993 AAAI symposium on “Games: Planning and Learning,” [Games-93], contains a wide spectrum of some of the most recent research.

A good survey of some decision-theoretic methods for control of computation is provided in [Dean-91]. The papers [Russell-Wefald-91] and [Russell-et-al-93] both contain very useful sections on the history of limited rationality in AI and elsewhere, with a number of good pointers to the literature. The working notes of the 1989 symposium on AI and Limited Rationality, [AILR-89], is a good selection of fairly recent work on limited rationality within the AI community, although it is difficult to obtain.

The body of literature in AI that is relevant to the topic of Chapter 4, “Abstract State-Space Search,” is large, and not adequately explored in that chapter. The Ph.D. thesis [Knoblock-91] would probably a good starting point for a more extensive search of that literature.

Finally, some of the ideas explored in this thesis concern fundamental technical issues of probability theory. In §2.11 and chapter 3 I make some proposals which would require a very deep knowledge of the literature in probability and statistics in order to properly evaluate. (For example, a working knowledge of measure theory such as in [Adams-Guillemin-86] would be very useful; see [Good-62].) Unfortunately I am not prepared at present to point to the right areas of related work in the statistics community, since my research into that literature is still very incomplete.

More details about related work will be discussed within each of the individual chapters.

Chapter 2

Evolution of Computer Game Playing Techniques

In order to understand the recent developments in decision theory and its application to computer game playing as a testbed AI problem, it will help us to review some of the past development of computer game-playing techniques. But before we begin describing the classical game playing algorithms, we will first take a look at the motivating task of *computer game playing* itself, and explicate some of the fundamental simplifying assumptions about this task that the field of computational game theory has traditionally made.

People often simplify problems in order to solve them. However, as AI researchers we need to simplify more than usual, because it's our not-so-bright computer programs that must do the solving. One classic AI simplification of a problem domain, exemplified in the early game-playing work discussed in §2.2 (p. 20), is to characterize the domain in terms of a well-defined set of possible states and state transitions. For those problems in which this simplification is reasonably appropriate, it facilitates automated reasoning by giving our programs a well defined, formalized space within which to search for a solution to the problem.

Unfortunately, if the domain is at all interesting, we often find that the sheer size of the state-space overwhelms the computational resources available to us. But in such cases, we'd prefer not to altogether abandon the comfortable simplicity of state space search. Instead, we compromise by searching only a part of the space. (See §2.3.2, p. 26.)

Partial search raises two obvious issues: what part of the space do we search, and how do we use information about only a part of the space when an ideal solution would require searching the whole space? The traditional answers to these questions have been kept as simple as possible; for example, by searching all and only those states within d

transitions of the initial state, or by making crude assumptions that just use the partial space as if it were the whole space (cf. minimax, §2.4).

Recently, however, researchers have been becoming more brave. They are beginning to abandon the simplest methods, and instead look for ones that can somehow be justified as being *right*. To do this, many are turning to the methods of probability and decision theory. The former can tell us what to believe, the latter tells us what to do. Thus we can answer the questions of what part of the space to search, what to believe as a result, and what external action, finally, to do. In §2.5 and §2.7 we discuss the advent of the probabilistic and decision-theoretic methods, respectively, in computer game playing.

Progress in being made steadily along these lines in computer game-playing, but researchers are beginning to be forced to confront problems with both decision theory and with the old partial state-space search paradigm. Some of the innovative ideas that are being developed to solve these problems will be discussed and expanded upon in chapters 3 and 4.

2.1 Why Games?

This thesis claims to be about developments in decision-theoretic AI in general. So why are we spending a large chapter studying DT methods in the context of game-playing, in particular? Perhaps we should concentrate instead on efforts to write decision-theoretic programs that actually do something useful. CGP researchers are constantly faced with such questions. Playing games is such a frivolous pursuit that it would seem that any research whose overt aim is to write programs to play games must be frivolous as well. Indeed, if the only aim of a piece of CGP research is to write a game playing program, then I would agree that that piece of research was frivolous.

Serious CGP research, in my opinion, must demonstrate an intent and desire to eventually develop general methods that can be applied to other problems besides games, or at the very least, to gain practice and experience in the use of general tools and techniques (for example, probability, decision theory, and statistics) which will then help us when we later attack more practical problems. Still, this begs the question of why we should pick game playing as the domain in which we will develop new general methods or exercise our skills.

To me, the reason is that a game is a clear, well-defined domain that may yet involve subtle issues of reasoning, and pose a real challenge for AI programs. Unlike "toy" domains made up especially for the programs built to solve them, games played by humans were not devised with the strengths and weaknesses of current AI methods in mind, so the game may involve more hidden subtleties and difficult problems than would the toy domain.[1]

But, unlike more general "real-world" situations, most games have a limited, finite specification in terms of their rules, which allows AI programs to at least have a fair chance at addressing them adequately, whereas in less formal real-world domains it is much harder to describe and reason about all the factors that might be relevant in reasoning about the domain. The domain may be too open ended for this to be feasible.

And yet, in games there are clear performance criteria: does our program win the game? What score does it obtain on average? How good is it compared to other programs that play the same game? These criteria help to tie the research to reality; if the program does not play well, then this suggests that the theory behind the program might be inadequate,[2] and the degree to which this is true can be quantitatively measured by, for example, the fraction of games won against a standard opponent. Games allow us to quantify the success of our ideas about automated reasoning.

Finally, the time limits imposed in some games encourage us to make sure that our algorithms not only reason well, but also that they reason efficiently. Efficiency is a vital concern for AI that hopes to be useful in the real world, because in the real world we do not have unlimited computing resources, and we can not necessarily afford to wait a long time to get an optimal answer, when instead we might be able to get a good enough answer from a less ambitious but much faster program.

---

1. However, we must be careful; if we have to work long and hard to find a game played by humans that our reasoning algorithm can handle, then perhaps this is no different than just making up our own "toy problem" after all—especially if we have to simplify the rules of the game before our method will work.

2. Although of course the program may also play poorly due to bugs or bad design in the implementation. Also, the converse is not necessarily true: if the program plays well at a particular game, this may be due to raw "brute-force" ability, or special-purpose engineering for the particular game, rather than being a result of general theoretical soundness. The degree to which this is true is not so easily quantifiable. However, this problem can be fixed. Later in this chapter we will discuss some new directions in CGP that aim to tie success in games more closely to general intelligence.

In summary, games, in general, are more approachable than other types of domains, and yet the right kinds of games can be very challenging from an AI standpoint, and offer many opportunities for the application of new, sophisticated reasoning techniques. Whether the existing CGP work has fulfilled the potential of games as a domain for AI is a separate issue that I will defer until later in this chapter.

2.2 Early Foundations

The formal analysis of games has had a long history. Games of chance have been studied using the mathematics of probability (and have historically been one of the major motivations for its development) for several centuries, but a general, systematic *game theory* really first arose with John von Neumann [vonNeumann-28], [vonNeumann-Morgenstern-44]. Von Neumann focused his attention on "games of strategy." His theory is most easily applied to games that have a formal, mathematical nature, as opposed to games having vague or ill-defined rules.

Von Neumann's work started a couple of different threads of further research. The book [vonNeumann-Morgenstern-44] was aimed primarily at game theory's relevance to problems in economics, and von Neumann's decision theory fit well into the economics tradition of attempting to model rational choices by individuals and organizations. Many of von Neumann's ideas concerning general decision theory were adopted and expanded upon by economists and by the newly inspired game theorists in the decades to come. Game theory was used in politics, economics, psychology, and other fields; the literature is too large to survey here.

Von Neumann himself had a significant background in computer science, and his game theory work was noticed and read with interest by computer scientists who were interested in demonstrating the power of computers to solve symbolic problems (e.g., [Shannon-50] and [Prinz-52]). Von Neumann's focus on strategic games was perfect for this, and his game theory provided guidance as to how to tackle that task in a systematic way that could facilitate implementation on a computer.

Thus, Claude Shannon set forth in the late 1940s to program a computer to play chess [Shannon-50]. Although Shannon mentioned von Neumann's basic idea of the *game-theoretic* value of a position, he did not make use of von Neumann's deeper idea of using maximum expected utility to make decisions in the presence of uncertainty. Perhaps this was because he felt that the kind of uncertainty present in chess was different from the kinds of uncertainty dealt with in von Neumann's decision theory. Chess is a deterministic game, having no chance element, so on the surface, decision theory's fundamental notion of expected utility does not apply. However, in chess, uncertainty about outcomes does exist, despite the determinism of the rules, due to a player's lack of complete knowledge of his opponent's strategy. Additionally, even if we assume a particular opponent strategy, such as perfect play, we lack sufficient computing power to calculate the outcome of a chess position under any such model. But regardless of the reason, Shannon did not draw any correspondence between his "approximate evaluating functions" and von Neumann's "expected utilities," a correspondence that only cropped up later in the development of computer game-playing, as we shall see in §2.5 (p. 38).

However, Shannon's work was paradigmatic, and has provided much of the context for the computer game-playing work since his time. Although his approach was more pragmatically oriented than von Neumann's theoretical treatment of games, Shannon's work depended to an equal degree on the well-defined nature of the domain (games of strategy) with which he dealt. Shannon used the concept of a branching tree of possible future positions linked by moves. The concept of searching such a tree, (or, more generally, a graph), of primitive states of the domain "world" was established, largely due to Shannon's work and the further computer-chess efforts by AI researchers that followed it, as a fundamental paradigm for CGP, which we will discuss in the next section.

Shannon's main contribution, utilized with by most of the game-playing programs to date with much success, was the idea of using a heuristic function that approximated the true (game-theoretic) value for positions a few moves ahead of the current position, and then propagating those values backwards, as if they were game-theoretic values, to compute approximate game-theoretic values of preceding positions. (We will see how this works in more detail in §2.4). Much criticism has been levied against this approach by later researchers, who point out that a function of approximate inputs is not necessarily a good approximation of the function with actual inputs ([Pearl-83], [Abramson-85]). We will go into more detail on these and other criticisms of minimax in §2.4 (p. 31).

Finally, largely overlooked for many years were some salient suggestions by Shannon as to how his algorithm might be improved by intelligently selecting which lines of play to consider, and how deeply to pursue them. Shannon's ideas in this area were preliminary, but he did have some very concrete suggestions that could have been implemented in the chess programs of that era, but were not (to my knowledge). The idea of *selective search* was finally developed in detail and incorporated into some actual programs several decades later; this work will be discussed further in §2.6 (p. 66).

But again, practically all of the later CGP work (to date), including the selective-search work, has been based on what I will call the *Shannon paradigm,* of so-called[1] "brute-force" search through a partial graph of primitive world-states. The next section will delve a little more deeply into the assumptions inherent in this way of attacking the problem, assumptions which we will consider later in the thesis (in chapter 4) how to abandon or weaken.

Other early work includes a chess program devised and hand-simulated by Turing around 1951 (described in [Bates-et-al-53]), a program at Manchester University for solving mate-in-two problems [Prinz-52], and Samuel's early checkers learning program [Samuel-59].

2.3 Partial State-Graph Search

In this section we examine in more detail the nature of Shannon's approach to game-playing and its impact on AI research in general. We break down the approach into its ontology (the way it views the world), its epistemology (the way it handles knowledge about the world), and its methodology (what it actually does, in general terms).

2.3.1 State-graph ontology

An intelligent agent, real or artificial, generally performs some sort of reasoning about a domain, a part of the world that is relevant to it or to its creator. An important characteristic of an agent is the *ontological stance* it exhibits towards the domain, by which I

1. Even by Shannon, in [Shannon-50].

mean the agent's conceptual model of how the world is structured and what kinds of things exist in it.

The ontological stance in the Shannon paradigm, one that a very large number of AI systems take towards possible circumstances in their domains, is to view the world in terms of its possible *states*. Typically, not much is said about what is meant by a world state; the term is usually left undefined. Here we will take a more careful look at what a world state typically is, and how that differs from what it could be.

The things that are usually called "states" in AI might be more precisely termed *concrete microworld states*. By "microworld" I just mean the domain, the part of the world that the agent cares about, or is built to deal with. A "microworld state" is a configuration of that microworld at some point in time. Finally, "concrete" means that the meaning of a given configuration is fully specific; it is complete and precise as to every relevant detail of the microworld at the given time; it is not at all abstract, nor can it be viewed as encompassing numerous possibilities. It is a single, primitive possible configuration. This is to be contrasted with the view of *abstract* states taken in chapter 4. If we choose to think of our microworld as being described by a definite set of variables, then a concrete microworld state is a complete assignment of values to those variables, rather than a partial one.

Hereafter, when I use the word "state," I will mean concrete microworld state, except where indicated otherwise. Although "concrete microworld state" is more descriptive, it is longer and less standard; thus we adopt the shorter term.

Not only does the use of states imply some sort of concept of time, with the microworld typically thought of as occupying a definite state at any given time,[1] but also, time is typically discretized to a sequence of relevant *moments*, with the world changing from one state to another between them. Additionally, there is often a notion of a set of the legal, possible, or consistent states that the world could ever occupy; states in the set are "possible" while there might be other describable but "impossible" states that are not in the set. This set is referred to as the microworld's *state-space*. (For example, in chess, we might

1. This can be contrasted with, for example, the Copenhagen interpretation of quantum physics, in which the world may occupy a probabilistic "superposition" of states. Interestingly, however, there is a lesser-known interpretation of quantum physics, Bohm's interpretation, that retains all the same experimental predictions while preserving the idea of the existence of an actual (but unknowable) world-state (see [Albert-94]).

consider unreachable positions to be states, but not to be in the chess state space. However, the idea of states that are not in the state space differs from some standard usages of the term "state space," and so I will not make use of this distinction much.)

Additionally, there is typically a concept of possible changes of state, expressed in terms of primitive *state transitions* (or "operators" or "actions") that are imagined to transform the world from one state to another between the discretized time steps. There is imagined to be a definite set of allowed or possible transitions from any given state.

The notion of a state space, together with the notion of state transitions, naturally leads to a conceptualization of domain structure in terms of a *state graph*, in which the nodes represent concrete microworld states, and the arcs represent the allowed state transitions. (See figure 1.)

Many of these elements were present in von Neumann's treatment of games, and indeed, they formed a natural analytical basis for the sorts of "strategic games" he was considering. Shannon, in developing computer algorithms for a typical strategic game, chess, continued to make use of this natural ontology for thinking about the structure of the game-playing problem.

And indeed, the vast majority of the AI programs written for game-playing domains, for similar types of domains such as puzzle-solving, and even for some supposedly more practical planning and problem-solving domains, have adopted the state graph, or its specialization the *state tree*, as their primary conceptualization of their domain's structure. State-graph ontology has proved to be a natural and fruitful framework in which

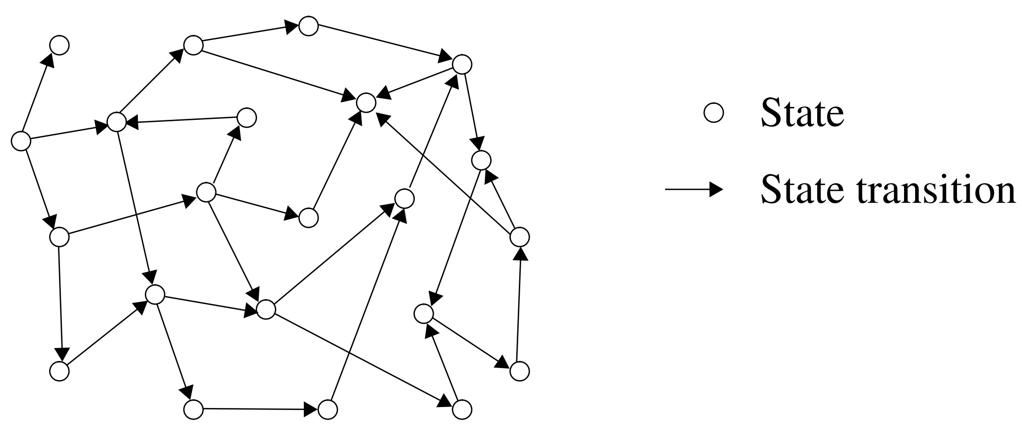


Figure 1. A state-transition graph

to develop well-defined and well-understood algorithms for reasoning in such domains. This success is no accident, since many of these domains are artificial ones that were defined by humans in terms of well-defined states and state-transitions in the first place. Humans, too, seem to make use of the state-graph conceptualization when performing deliberative decision-making in these domains.

However, I will argue in chapter 4 that, even within these state-graph-oriented domains, the state-graph conceptualization is not the only framework that humans use to support their reasoning, and that certain other models permit a greater variety and flexibility of reasoning methods. I propose that the development of AI programs capable of more humanlike and general reasoning might be facilitated by research on programs that demonstrate how to use such alternative conceptual models to exhibit improved, more humanlike performance, even when applied to simple game and puzzle domains. I am not the first to promote this idea; we will mention some of the existing efforts along these lines in chapter 4.

In any case, this "state-graph ontology" is the part of the Shannon paradigm that has perhaps had the most influence on computer game-playing, and on AI in general. It is so pervasive that it is rarely mentioned, and yet it shapes our ideas about AI so strongly that I believe we would be well-advised to occasionally explicate and reconsider this part of our common background, as we are doing here, and at least take a brief look to see if we can find useful ideas *outside* of the state-graph paradigm, ideas that we may have previously missed due to our total immersion in the language of states and transitions. For example, in some of the theoretical directions considered in chapter 3, we will refrain from adopting the state-graph ontology, in favor of a more general view.

Although state-graph ontology gives us a structure within which we can analytically describe what the problem is that we are trying to solve, and what an ideal solution would consist of, it still does not tell us what facts about the domain would be known by a real agent that we design. This is the role of an epistemology, and specifically, we now examine the *partial state-graph* epistemology, another major aspect of the Shannon paradigm.

2.3.2 Partial state-graph epistemology

Besides an ontology, another important characteristic of any agent is the scope of its knowledge about the domain in which it is working. In particular, if we are told that an agent models a domain in terms of a certain kind of state-graph, this still does not tell us very much regarding what the agent does or does not know about the possible circumstances that might occur in its microworld.

Given the state-graph ontology described in §2.3.1 (p. 22), it is natural to describe agents' states of knowledge about particular domain circumstances in terms of an *examined partial graph*, that is, a subgraph of the domain's state-graph expressing that set of nodes and arcs that the agent has *examined*. We will say a node or arc has been examined, or "searched," when an agent constructs a representation for it. Generally, an agent's purpose in examining part of a state-graph is to infer useful information from it. An agent might know things about some of the nodes that it has not yet examined; for example, it may have proven that all nodes in an entire unexamined region of the graph have a certain property.[1] Nevertheless, we will still treat the examined partial graph as a special component of an agent's knowledge, because there is usually a lot of information about a state

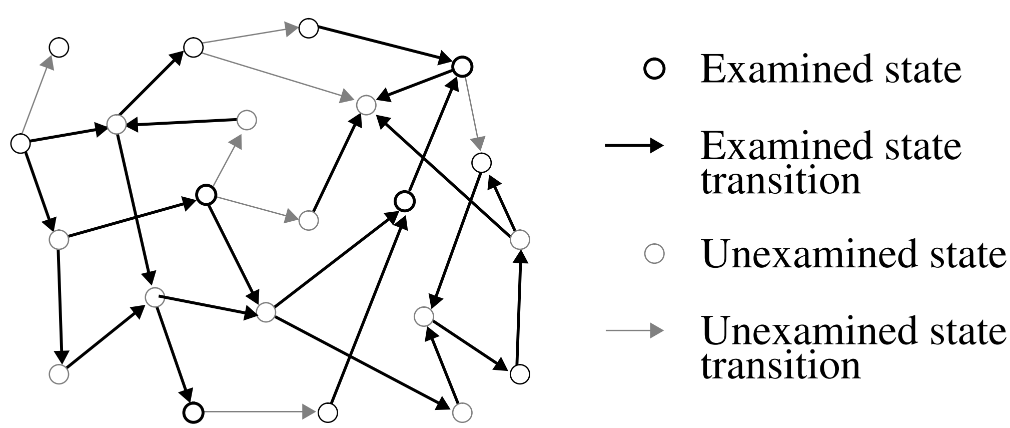


Figure 2. A partial state-transition graph. In general an arbitrary subset of the states and transitions may have been examined (internally represented at some time).

---

1. A concrete example: the alpha-beta algorithm performs game-tree pruning by proving (under certain assumptions) that none of the nodes in an entire unexamined subtree will occur in optimal play.

that an agent can only obtain, as far as we know, by examining a corresponding region of the state-graph.[1]

For some small game and puzzle domains (e.g., tic-tac-toe and the 8-puzzle) the domain's state-graph is so small that a program can examine it exhaustively in a short time. In these domains, we can process all the information needed to determine an ideal solution or move choice very quickly, and so there is not much to say about epistemology.

More interesting are the larger games, puzzles, and strategic situations, in which the state-graph may be far too large to examine in a feasible time. It may even be infinite. In such domains, the particular state-graph epistemology we apply to our agents becomes an important consideration in their design, and this epistemology can have a major influence on the agent's performance. If performance is to be at all acceptable, the examined partial graph cannot be the full graph, so the knowledge that will be available to our agents at a time when they must make a decision will depend strongly on the agent's earlier choice of which of the many possible partial graphs to examine.[2]

Shannon's answer to this question for chess was the subsequently-popular *full-width search policy,* which says to look exactly *d* moves ahead along every line of play, for some depth *d*. However, Shannon realized that this was an arbitrary policy, and not, in any sense, the "right" way to answer the question, and so he proposed a couple of more sophisticated approaches. Later developments in CGP that expanded upon these suggestions are discussed in §2.6 and §2.7. However, Shannon's suggestions and the later developments were still largely *ad hoc*. More recently, partial-search epistemology has been tackled from a decision-theoretic standpoint; these efforts are discussed in §2.8 (p. 69).

So, once we have decided on this sort of epistemology, that our agent's knowledge will consist (at least in part) of knowledge about portions of the state-graph representing the domain's structure (as viewed in our ontology), we have a general guiding framework for the design of our reasoning algorithms, at least ones intended for strategic games and

1. For example, the only way we know to obtain the true game-theoretic value of a general chess position is to examine a large tree of successor positions.

2. Moreover, we note that this choice of partial graph obviously cannot be made through explicit consideration of all possible partial graphs—for that would take exponentially longer than examining the entire original graph! Some higher-level policy for choosing the partial graph is needed.

similar problems. But there remains the question of our agent's more detailed, pragmatic behavior; exactly how will its decision-making proceed?

2.3.3 Partial state-graph methodology

We are led to two crucial methodological questions for agents that are built under the presumption of a partial state-graph epistemology. Each agent should, either in its design or at run-time, answer these questions:[1]

- What partial state-graph should I examine?
- How shall I derive useful knowledge from my examination of this partial graph?

I submit that the ways a program answers the above two questions are useful dimensions along which to categorize AI programs that use the partial state-graph paradigm. I suspect that the computational effort expended by these programs can be naturally broken down into computations that are directed towards answering one or the other of these questions.

Many programs answer these questions incrementally, that is, they start with a small partial graph, and examine more nodes and arcs slowly, while simultaneously deriving information from the intermediate graphs, and using this information to guide the further growth of the graph. This can be contrasted with the alternative of creating the whole partial graph in one large atomic step, and only then stopping to derive information from it.

Often the goal behind the incremental approach is to provide an *anytime algorithm* [Dean-Boddy-88] that can be stopped at any time and asked to immediately provide an answer based on the partial graph examined so far.[2] (Anytime algorithms have been investigated as a general solution to some of the problems of limited rationality, for example, see [Zilberstein-Russell-93, Zilberstein-93]. This work will be discussed briefly in chapter 3.) Another reason for the incremental approach is that if an algorithm processes states as it examines them, then it may be able to save memory by expunging most of the examined

1. [Newell-et-al-58] provides a similar characterization of the Shannon paradigm.
2. For example, the iterative deepening approach in game-tree search, see [Newborn-89].

nodes, and only keeping the useful information that is derived from them.[1] In my terminology, I will consider expunged nodes to still have the status of "examined" nodes, unless all information gained from examining them is thrown away as well. For example, if, in game-playing, we throw away all the results of our search after making a move, then the nodes we had searched are now "unexamined" again, until and unless we revisit them in the course of choosing our next move.

Another rarely-mentioned but important characteristic of most partial state-graph search programs is that the examined partial graph is usually a connected graph (undirectedly). This connectedness is a consequence of the natural graph-examination scheme of always examining nodes that are connected to ones that have already been examined and stored in memory. The connectedness property also tends to make it easier to extract useful information from the partial graph, by using rules for propagating information along state transitions and combining information at nodes in a semantics-preserving fashion. We will see several examples of such algorithms. However, one can imagine a partial graph-search scheme that examines, for example, scattered randomly-chosen nodes, or that expands the graph outwards from a small selection of unconnected search "islands." (See figure 3.)

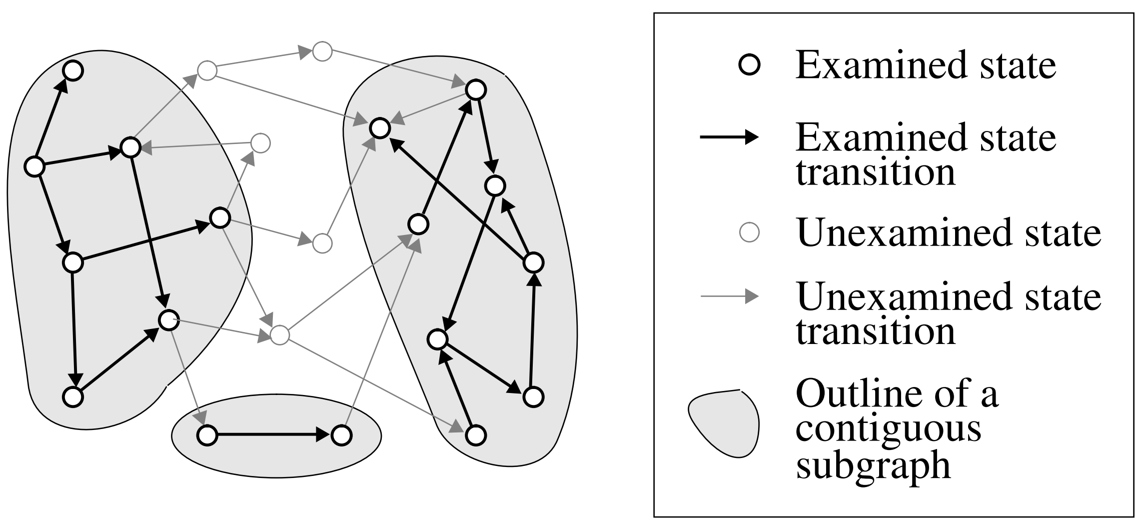


Figure 3. A partial state-transition graph with a few contiguous examined regions, or search islands. The examined graph has more structure than in figure 2.

1. This is done by alpha-beta, see §2.4.3, p. 37.

2.3.3.1 Choosing a partial graph to examine

The first fundamental methodological question is how to determine exactly what partial state-space graph we are going to produce. As mentioned above, this determination is usually done incrementally. However, methods exist that examine a predetermined set of nodes without regard to information obtained from examining them, for example, a depth-*n* full-width minimax search.[1] Most methods are incremental to some degree; for example, alpha-beta search uses the values of some subtrees to choose which other subtrees are worth searching. A* ([Hart-et-al-68], [Hart-et-al-72]) is a single-agent example of an incremental search.

2.3.3.2 Deriving useful information from a partial graph

The second fundamental question is how to derive information from a partial graph that will be useful in making a final decision. The most common technique is to obtain some information at the nodes located on the outer edge of the partial graph (i.e., at the *leaves*, if the graph is a tree), and then to propagate that information toward the current state, along the arcs of the graph, transforming the information along the way to determine how it bears on the overall decision or problem faced in the current state. We will see many examples of this sort of gather-and-propagate algorithm in the CGP research discussed in the sections ahead.[2]

The information at the leaves is usually obtained by some heuristic function, although the heuristic may be obtained from relevant statistics about similar past nodes, rather than being handmade by a human.

Certain propagation algorithms (e.g., minimax) can be interpreted as implicitly assuming that a piece of heuristic leaf information is a correct representation of some function of the entire unexplored part of the graph below that leaf (for example, the leaf's game-theoretic value). The probabilistic and decision-theoretic algorithms we will see later in this chapter attempt to weaken or abandon such assumptions.

---

1. Here I am not counting the set of children as information obtained from examining the parent.

2. However, not all state-space CGP techniques use information solely from nodes on the periphery of the examined partial graph—some also utilize information from interior nodes; for example, see [Delcher-Kasif-92] and [Hansson-Mayer-89]. Whether this is the "right thing" to do is still unclear.

A final simplifying assumption is that many algorithms for extracting information from partial graphs assume that the graph has a restricted structure, such as that of a tree or a DAG. The reason for this assumption is often that the correct algorithm would be too slow or too hard to describe. Such assumptions may even be used in cases when they are known to be false, under the hope that the inaccuracies caused by the false assumptions will not degrade decision quality very much. In practice this may be true, but it makes the theoretical performance of the algorithm much harder to analyze.

2.3.4 Is partial state-graph search really what we want?

In the above subsections we have revealed some of the simplifying assumptions about the world, and how our agents should deal with it, that are implicit in the paradigm established by Shannon and much early AI work. To summarize:

- The history of the world is broken into a discrete sequence of times.
- At any given time, the world is fully characterized by the unique state it occupies at that time.
- There is a definite space of possible states the world may be in.
- There is a definite mapping from states to possible next states.
- Our agents will represent the world by a graph of nodes representing these world-states, with arcs representing possible transitions.
- Our agents will reason about the world by examining a subgraph of this state-graph.
- Our agents will explore contiguous regions of the state graph.
- Our agents will make approximate guesses about properties of the graph beyond the part that has been examined.
- Our agents will infer what to do by "propagating" information through the graph, typically from the edges to a known "current" state.

This is really quite a large number of things to assume without justification. And yet, much research using these assumptions, such as most of CGP work, proceeds without even a mention of any justifications, either because the researchers are not even realizing they are making these assumptions, or because they presume that the reasons for making these assumptions have already been adequately established in the literature, and they are just following along in the tradition of the field.

However, it is my opinion that the field has *not* adequately established why these assumptions should be adopted, and that this paradigm is being followed mostly just due to its own momentum. The fact that it is used so universally means that it doesn't even occur to most researchers to question it.

This thesis, however, intends to question some of these assumptions (and any others encountered), and either find satisfactory justifications for them or present better alternatives. The later chapters will dive more deeply into this enterprise of questioning, but for now, let us just accept that partial state-graph search is the basis for much of the current work in AI, and CGP especially, and see what we can learn by examining in detail a number of the existing CGP techniques, while revealing and questioning some of their additional assumptions along the way.

2.4 The Minimax Algorithm

The previous section described the crucially important, but rarely-mentioned epistemological assumptions that are pervasive throughout all computer game-playing research, and much of the rest of AI, from Shannon up to the latest work. Now we shall go into more specific detail about Shannon's algorithm, and some of the particular assumptions under which it can be justified, even beyond the basic assumptions and limitations of the partial state-graph search paradigm. Together all these assumptions make up what I call "the Shannon paradigm."

The Minimax algorithm has been described and explained informally too many times to repeat that effort here. Readers unfamiliar with the algorithm should refer to any introductory AI textbook (such as [Rich-Knight-91] or [Winston-92]) for a tutorial explanation. Instead, my contribution here will be a more formal treatment that will be suited to our later discussions.

2.4.1 Formalization of the minimax propagation rule

We assume the game we are considering has two players, which we will refer to as "Max" and "min," and can be represented as $G = (\Omega, \delta, M, v)$, where Ω is the space (set) of possible states the game may be in, $\delta: \Omega \to 2^{\Omega}$ is the *successor* or *transition* function mapping each state in Ω to the set of its possible successor states, $M \subseteq \Omega$ is the set of states in which Max gets to determine the next state (whereas min gets to determine the next state in $\Omega - M$), and $v: \{s \in \Omega: \delta(s) = \varnothing\} \to \mathbf{R}$ is a function that maps each terminal game state (where a terminal game state is one with no successors) to a real number, with the intention that the number expresses the state's degree of desirability from Max's point of view. It is also assumed that $-v(s)$ expresses the degree of desirability of s from min's point of view.[1] (Later in this section we will question some of the definitions and assumptions in this framework; for now we just present it.)

Then, assuming the range of δ includes only finite sets, we can define the *game-theoretic value* $V(s)$ of any $s \in \Omega$ recursively as follows:

$$V(s) = v(s), \text{ if } \delta(s) = \varnothing\text{; otherwise,} \tag{1}$$

$$V(s) = \max_{t \in \delta(s)} V(t), \text{ if } s \in M\text{; and} \tag{2}$$

$$V(s) = \min_{t \in \delta(s)} V(t), \text{ if } s \notin M. \tag{3}$$

We note that if there are directed paths of infinite length in the graph (Ω, δ) (in other words, if the game may go on forever), then there may not be a unique solution to these equations for all nodes.[2] However, in practice, for example in a program for calculating $V(\cdot)$ of all nodes, we could handle this case also, by declaring infinite length games to have value 0, corresponding to a "draw," so long as the game graph (number of states) is not infinite. A detailed algorithm for this is provided in §2.5.4.4 (p. 60). But for now, we will, for simplicity's sake, restrict our attention to only those cases where a unique solution does exist.

---

1. In other words, we are assuming that the game is "zero-sum."

2. It is often said that the "50-move rule" in chess implies that the game is finite. But actually, the rules of chess do not actually require either player to claim the draw; so in principle a legal chess game could indeed continue forever.

Clearly, if we have access to only a partial graph, then we cannot in general compute $V(s)$ even for s in the partial graph. If all the successors of a node s are in the partial graph, and we can compute $v(t)$ for all successors t that are terminal, then we can compute $V(s)$. But what do we do for nodes whose successors have not all been examined? Shannon's answer is to compute, for each "leaf" node l (a leaf is an examined node whose successors are all unexamined), an *estimate* or *approximation* $\hat{v}(l)$ of the game-theoretic value $V(l)$ of l. Then, if we assume that all children of non-leaf examined nodes are also examined, we can just use (1)-(3), putting hats on all the v's and V's, to compute $\hat{V}(e)$ "estimates" for all examined nodes e. If the leaf estimates are correct, that is, if $\hat{v}(l) = V(l)$ for all leaves l, then it will also be the case that $\hat{V}(e) = V(e)$ for all e; i.e., the estimates of all non-leaf examined nodes will be correct as well. Analogously, it is hoped that if the $\hat{v}(l)$ are, in some sense, "good" approximations to the game-theoretic values, then the derived estimates will be good approximations as well. (Whether this hope is borne out will be examined in the next few subsections.)

So, this is how the minimax rule propagates information through the examined partial graph. In order to actually use minimax to decide on a move when it is our turn, we assume that we have determined the set $\delta(c)$ of immediate successors of our current state c, and that we have examined a partial graph including those successors and have computed their minimax values according to the above propagation rule. Then, if we are taking the role of player Max, we choose the successor (move) s that has the highest $\hat{V}(s)$, and if min, we choose s so as to minimize $\hat{V}(s)$.

2.4.2 Basic assumptions

Minimax involves assumptions that go far beyond the basic elements of the state-space paradigm described in §2.3. First, there are the basic assumptions having to do with the domain's being a strategic game: the state-space contains terminal states, and the desirability of the entire episode of interaction between the agent and its environment is reduced to the desirability of the final game state reached. Moreover, the game is assumed to be a zero-sum, two-player game. The game is deterministic, with the choice of state-transition always being completely made by one of the two players (and not involving any nondeterministic external factor).[1] Minimax requires that the successor function and the value of terminal states should be easily calculable, and that the number of successors of nodes not be too large to enumerate. Finally, using minimax requires that the game be one of "perfect information" where we can identify, whenever it is our turn, which state the world actually occupies.

However, even apart from these assumptions, one fundamental problem with minimax is that there is no theoretical justification for why it should work! Although the minimax rule is, by definition, the correct propagation rule for game-theoretic values, there is no theoretical reason to believe that, when given just "approximations" to game-theoretic values of leaf nodes, minimax will produce any reasonable approximation to the game-theoretic values of nodes higher in the partial tree. In fact, as we will see in §2.4.4 (p. 38), under some theoretical models of what kind of "approximation" the evaluation function $\hat{v}(\cdot)$ is providing, we can show that minimax can actually *degrade* the quality of the approximation, more so with each additional level of propagation. This is termed "pathological" behavior.

Nevertheless, we can attempt to justify minimax by showing that it does the right thing under certain sets of assumptions, and arguing informally that those assumptions are likely to be "almost" correct "most of the time" in practice, or that if the assumptions are categorically *not* correct, then the error incurred by this incorrectness is likely not to matter much. However, it should be kept in mind that such arguments are informal and inconclusive, and would probably be ignored if it were not for the fact that the Shannon paradigm has been empirically so successful.[1] One hopes that some of these informal arguments could be turned into formal approximation theorems that show, for realistic models of games and evaluation functions, that minimax is close to being correct most of the time, in a formalizable sense. However, no such analyses have been performed, to my knowledge.

Let us look at some of the sets of assumptions under which minimax can be shown not to incur errors. The first and most obvious one is just the assumption that the evalua-

1. If the desirabilities behave like utilities (i.e., if our goal is to maximize our mathematical expectation of them) then it is easy to add a rule to minimax propagation to accommodate games having nondeterministic "chance" moves as well; but the large body of more sophisticated algorithms (such as alpha-beta pruning) that are based on the Shannon paradigm will not so easily accommodate this addition.

1. Again, see practically any general AI textbook or general survey of computer game-playing, such as [Newborn-89].

tion-function values are the correct game-theoretic values. Then, of course, minimax yields correct values for all nodes. However, if we really want to assume that our evaluation function gives game-theoretic values, then we need not search at all, and so although it yields a correct answer, doing a deep minimax evaluation does not really make sense under this assumption. One can try to save the argument by saying that the evaluation function is assumed to be correct deep in the tree, but may be prone to errors closer to the root, but if one is not careful in one's model of this error gradient, pathological behavior may still occur ([Pearl-83]).

An alternative set of assumptions goes as follows. We interpret evaluation function values as giving our current expected utility (e.g., in a 2-outcome game, just the probability of winning) if play were to proceed through that node, and then we assume that our opponent shares those same assessments of expected utility, and that both he and ourselves will continue to have those same assessments while play proceeds, until one of the leaf nodes is reached. Assuming that both sides will choose moves they perceive as having maximum expected utility, this allows us to predict, with certainty, the value of the move that would be made at each node (although not the move itself, since there might be more than one move that yields maximum expected utility). I claim that under these assumptions, the minimax propagation rule yields consistent assessments for our expectation of the utility of the examined nodes.

This set of assumptions is an instance of a case where we might persuasively argue that the assumptions are close to being correct, or that their incorrectness does not matter so much. In an informal sort of way, they are conservative assumptions, in the sense that they assume things are worse for us than is actually the case. How is this so?

First, if we assume that our assessments of leaf values will not change as play proceeds, that is a conservative assumption, because it means we will not gain any new information about the leaf values beyond what we have so far, which is worse than the truth. If we assume that the opponent knows everything that we know, then that, too, is a conservative assumption. More conservative would be to assume the opponent knows a lot more than we do, but we could not hope to model such knowledge,[1] so we might as well pro-

1. Not everyone would agree with this argument. [Carmel-Markovitch-93] attempts to do just this sort of modeling.

ceed on the assumption that the opponent knows everything we know, but no more.[1] Similarly, if we assumed that the opponent were to gain more information about leaf values during play (and that we would not), this would also eliminates all hope of success, so we stick with the assumption that he will continue to just know everything that we know now. Finally, since the game is zero-sum, assuming that the opponent will act to maximize his expected utility is a conservative assumption, whereas the assumption that *we* will continue to attempt to maximize our utility, while not as conservative as assuming we won't, is at least known to be true.

So, although this argument is completely informal, it perhaps lends some insight into how to explain the high level of performance of minimax and its derivatives in practice: in a sense, minimax makes *conservative* assumptions, and thus serves as a good "engineering approximation." However, the assumptions are still rather questionable, and the fact remains that the only existing theoretical justifications of minimax depend on these assumptions. It would be better to have a propagation algorithm that did the right thing under assumptions that were more clearly realistic. Some attempts to do this are described in sections §2.5 and §2.7.

2.4.3 Alpha-beta pruning

Alpha-beta (α-β*) pruning* is a well-known technique for speeding up minimax search. (See [Rich-Knight-91] for a tutorial explanation.) Its exact origin is unknown; [Newell-et-al-58] briefly mention a technique they use that sounds similar, but do not give sufficient detail to tell if it is precisely the same algorithm. The first known full description of the algorithm is [Hart-Edwards-61], where one form of it is attributed to McCarthy but no reference is given. Some experiments on the algorithm's performance are described in [Slagle-Dixon-69], and some formal analyses of the effectiveness of the algorithm appear in [Fuller-et-al-73] and [Knuth-Moore-75]. (α-β is sometimes wrongly attributed to Knuth in the CGP literature; we wished to avoid repeating that mistake here.)

1. This is an instance of the general observation made by Doyle and others that it can be rational to adopt a belief, or at least act as if you believed it, even if (probabilistically speaking), you don't think it's likely to be true.

The α-β algorithm has received much attention in the computer game-playing literature. However, it is not of very much interest to us here, for several reasons. First, it does not lend us any new insights as to the meaning of node values or how to propagate them; it is merely a speed-up algorithm that computes the same value for a game tree that minimax would, but without examining all of its nodes. α-β does, however, serve as an example of how to incrementally utilize results from a partial search to guide the choice of nodes to examine in further search, a concept we will see more examples of later.

Second, although some authors have made much ado about how alpha-beta is "provably correct," what we should try to keep in mind is that, although the values it computes are indeed correct minimax values for the equivalent full-breadth partial tree, they are only correct, in any deeper sense, in the same sense that minimax is correct, i.e., only under the rather questionable assumptions mentioned in §2.4.2. There is still not much that can be said, theoretically speaking, about why a deep alpha-beta search (equivalent in its output to a full-width minimax search of the same depth) should actually lead to good game-playing decisions.

2.4.4 Pathology

Earlier (§2.4.2) I mentioned that under some attempts to model exactly what it is that an "approximate" evaluation function is doing, it is possible to prove that propagation via the minimax rule can actually degrade the quality of the approximation, rather than improve it. This indicates that, in some cases, deep searches may be futile. In this section I discuss these analyses in more detail, because they provide good examples of how to assign a clear semantics to node values. The move towards providing a clear node-value semantics is a vital step in the attempt to make sense out of game-playing algorithms and synthesize a "correct" approach. The probabilistic and decision-theoretic techniques discussed in §2.5, §2.7, and §2.8 similarly require some sort of well-defined semantics for node values.

The first exposition of pathology was in Ph.D. work by Dana Nau (see [Nau-80] for a summary). In Nau's model, evaluation function (EF) values are interpreted as being independent noisy measurements of nodes' true game-theoretic (GT) values. In other words, for each possible GT value, we have a probability distribution over what the EF value of a node with that GT value would be. We assume that the EF values for different leaf nodes are chosen independently according to these distributions.

Additionally, in Nau's original model, the distribution for a win node was symmetrical to that for a loss node, and the game trees had a restricted structure, but later it turned out that these properties of the model were not important for the pathology results.

Nau proved, within his model, that as search depth increases, the probability of choosing a game-theoretically optimal move using minimax (at nodes where both optimal and suboptimal moves exist) approaches the fraction of moves that are optimal, i.e., it approaches the probability that a random choice would yield an optimal move. These results were rather surprising, because in actual game-playing programs, increasing search depth has been empirically found to achieve higher move quality.

Nau's results were generalized in several ways by Pearl in [Pearl-83]. Pearl showed that for game trees of uniform structure (but less restricted in structure than in [Nau-80]), a very high rate of improvement of evaluation function accuracy with depth was needed to avoid pathology. But under a certain model of nonuniformity in the game tree, in which terminal positions are distributed randomly, pathology is more easily avoided, possibly explaining why real games do not seem to exhibit the phenomenon.

In [Nau-82], Nau investigates another way of explaining the lack of pathology in real games, via considering dependencies between sibling nodes in the game tree. In contrast, the model of evaluation function error used in the previous pathology results assumed that EF values were independent noisy measurements of GT values, even for nodes very close together in the game tree. One might suspect that in real games like chess, where most characteristics of a position only change incrementally, EF and GT values of neighboring nodes would be more closely related. Thus, Nau investigated a couple of "actual games," one in which the GT values exhibited such correlations, and one where they did not, and showed that pathology was avoided in the case where dependencies existed. However, the "board-splitting" games investigated were somewhat unrealistic. Not only did they *have* perfectly uniform branching-factor, uniform-depth game trees (unlike, e.g., chess) but also the game consisted *only* of a game tree—there were no high-level "rules" giving structure to the arrangement of game outcomes, other than the correlations Nau introduced at random. Moreover, the evaluation functions "cheated," in that they essentially examined the entire game tree below a node to determine its value, whereas in real games, the whole point of the EF is that the game tree too large to examine. A final problem with the board-splitting model is that it precludes investigating games having large numbers of moves, because the representation of a game position is essentially just a game tree, and so it is exponentially large in the number of moves left in the game.

Thus, the board-splitting game really is no more of an "actual game" than are the abstract, idealized game-tree models to which it is isomorphic. So it is somewhat misleading to draw conclusions about typical real games from results in the board-splitting domain. Further investigations after Nau's (e.g., [Abramson-85]) have continued to draw such conclusions, but I propose that in the future it would be better to just openly admit that the board-splitting "game" is really just an idealized game tree, and reserve the term "actual game" for games having more structure. It would be interesting to see pathology demonstrated in such games. In §2.5.4.5, I study one real game exhaustively, and show that pathology does occur in it (up to a point).

An interesting recent paper on pathology is [Delcher-Kasif-92]. They use the same model of evaluation function error, but show that if interior node values as well as leaf values are taken into consideration, the effects of pathology are reduced.

All the work in pathology has in common a certain semantics for evaluation function values, namely, they are treated as if they were noisy probabilistic measurements of the node's true game-theoretic value. However, for most evaluation functions, this semantics does not fit the actual situation exactly; the evaluation function is rather just some arbitrary function of position, and so there may be hidden interrelationships between the EF and GT values that the "noisy measurement" model fails to capture. But, as we will see later, it is possible to construct statistical evaluation functions that fit the probabilistic semantics more precisely.

2.5 Probabilistic evaluation

Partly spurred by the results on minimax pathology, researchers went on to consider alternatives to minimax that would abandon some of its assumptions. One intriguing interpretation of minimax node values we encountered in §2.4.2 was that node values represented our *expected* values for nodes, under our uncertain state of knowledge about what would happen next if the given position were actually reached. For a two-outcome (win/ loss) game, the expected value is just equal to our probability of winning, under, at most, a linear transformation, so hereafter we will just talk about the probability of winning instead. However, recall that in order for minimax to make sense under this interpretation of node value, we had to make some rather implausible additional assumptions. In particular, we had to assume that our opponent shared our assessments, and that both players would continue to maintain the same assessments until one of the leaf nodes was reached. Could this assumption be changed to something more realistic while still maintaining a relatively simple propagation rule?

2.5.1 Modifying minimax's assumptions

One obvious way to change the assumption is to go to the opposite extreme: instead of assuming that neither player gains any new knowledge until a leaf is reached, one can assume that instantly after the current move is made, both players will gain complete knowledge of the true outcome that will occur at each leaf node,[1] thus enabling both players to play optimally (with respect to that knowledge) after the current move choice, until a leaf is reached. This means that the quantity we need to calculate at each child of the root (current) node is the probability that it is a win with optimal play down to the leaves, given the probability of winning at each leaf.

Unfortunately, a probability distribution over each of several variables (in this case the outcome of each leaf) is not sufficient information to give a distribution over the joint combination of the variables, unless we know whether and how the variables depend upon one another. A parent of a leaf will be a win (for the player to move from the parent) if and only if one of the children is a win (since we are assuming that that player will know the win/loss status of each child, and thus will make a correct move from the parent). But this condition is a property of the entire configuration of all the children, not of the children

1. This is not necessarily the game-theoretic value of the node; it is just an outcome that we somehow know will occur (perhaps, for example, because we discover that we will reach a certain kind of end-game that we know one of the players does not know how to solve).

taken individually. So without some sort of model of the dependencies between siblings, we cannot properly propagate probabilities from children to parent.

However, we could conceivably calculate bounds on the probability of the parent being a win, where the bounds would encompass the entire range of possible joint distributions we might have for the children that would be consistent with our individual distributions over each of them. How would this be done? We can derive this as follows.

First, we note that the minimum win probability at the parent (for the player to move there) is equal to the win probability of the child with the highest win probability. This is because we know that we will definitely win if that child is a win (remember, we are assuming we will learn the true values of the leaves before any future moves). But our joint distribution could be such that if the child is a loss, all the other children are losses as well, so the minimum probability for the parent is no more than for that child. Thus, the lower bound on the parent is the max of the children.

What about the upper bound? Well, it could be that all the children win in disjoint circumstances, so that the total probability of any one of the children being a win is the sum of the probabilities of each of them being a win individually—up to, of course, the maximum probability of 1.

Thus, we can accommodate our lack of a model of the dependencies between the value of leaf nodes by propagating probability bounds up the tree in this way. However, we will still have to make a rather arbitrary decision about how to choose between moves with overlapping probability bounds at the children of the root. So this solution is rather unsatisfactory.

Instead, we can make a new assumption to model dependencies between leaf values, so that we can derive a joint distribution over the children of a node from our individual distributions for each of them. Perhaps the simplest and most obvious model is just to assume that there *is* no dependence; the outcomes at the leaves are independent. Is this a good assumption? We will attempt to answer that question in §2.5.4 (p. 47). In any case, this assumption yields a propagation rule that has received much attention in the literature on CGP in AI: the probability product rule.

2.5.2 The product rule

In the previous subsection we assumed that the evaluation function values of a leaf represents our expected utility if we were to go to that leaf, or equivalently (with scaling) our probability of winning (in a 2-outcome game). We also assumed, in direct opposition to minimax's assumptions, that as soon as we pick a child of the root to move to, both ourselves and our opponent will gain complete knowledge of the "true" value of each leaf (say, by expanding the leaves completely), i.e., each player would know whether or not we would win if we went there. This allows both players (assuming basic competence on their part) to play optimally down to the leaves, after which the player certain to win from the leaf eventually does. Under this assumption, the probability of winning from a child of the root is just the probability that a random assignment of leaf outcomes (chosen according to the probabilities at the leaves) would make that child a win.

Now, suppose we also assume that our joint distribution over all leaves is just the independent combination of our distributions for the individual leaves' outcomes. We want a propagation rule that will tell us, for each child of the root, what our probability of winning is if we go there.

The correct propagation rule for this model was first stated by Pearl ([Pearl-81], §3.5, [Pearl-84], §10.2.4), although he was not as explicit as this about what "probability of winning" meant, and seemed to be referring to a node's probability of being a game-theoretic win, rather than just being a node that the current players would happen to win from. (However, the two models are equivalent in terms of their propagation rule.) Pearl's probability propagation rule is as follows.

We assume we have an examined partial game graph $G = (\Omega, \delta, M, p)$, similar to the G we used in §2.4.1 (p. 32), but with the $v(\cdot)$ function replaced by a function $p\colon \{s \in \Omega\colon \delta(\Omega)=\varnothing\} \to [0, 1]$ mapping leaves of the partial graph to our conditional probability of Max winning, given that play proceeds to that leaf. Now, we define the function $P\colon \Omega \to [0, 1]$ (in place of our earlier $V(\cdot)$) recursively as follows,

$$P(s) = p(s), \text{ if } \delta(s) = \varnothing; \text{ else} \tag{4}$$

$$P(s) = 1 - \prod_{t \in \delta(s)} 1 - P(t), \text{ if } s \in M, \text{ and} \tag{5}$$

$$P(s) = \prod_{t \in \delta(s)} P(t), \text{ if } s \notin M. \tag{6}$$

Together, these formulae are referred to, for obvious reasons, as the "product rule." Although these equations work only for the simple case of a two-outcome game, they can be extended straightforwardly to a game with several outcomes.

Pearl introduced the product rule almost just as an aside to a discussion about why minimax works, and about the phenomenon of pathology. Pearl originally hypothesized that the product rule might not show pathological behavior due to its increased "statistical validity" over minimax. But it is important to keep in mind that the product rule is only valid *under the assumptions* described above, assumptions that do not seem very close to the conditions that obtain in a real game. As a result, the product rule displays artifacts of its own: win probabilities may fluctuate between very high and very low values as search depth increases. (I replicated these results in experiments on several simple games, in which leaf win probabilities were not derived from an ad-hoc hand-tuned evaluation function, or even from the results of learning from many games, but rather from the true counts of the number of nodes in various classes that were wins or losses, out of the entire game tree. Details of these experiments are provided in §2.5.4, p. 50.)

Despite the product rule's flawed assumptions, researchers took Pearl's proposal seriously and proceeded to perform various experiments comparing the product rule to minimax and checking for pathologic behavior (e.g., see [Chi-Nau-87]). In general, it was indeed found to avoid pathology.

But, as noted above, the assumptions necessary for the product rule to make theoretical sense, although different from those needed for minimax, fail to be self-evidently more realistic than minimax's. The product rule, in a sense, takes the opposite extreme from minimax, in terms of its model of the players' future beliefs. So perhaps a more realistic model would tend to interpolate between the two models. The model introduced by Baum in [Baum-92] and explored further in [Baum-Smith-93] and [Baum-93] attempts to do just that. Baum and Smith develop workable assumptions that *are* pretty clearly more realistic than those used by either minimax or the product rule, but the result is a significantly more complex model and algorithm, requiring more time to develop to the point where satisfactory amounts of empirical data can be collected to validate the model's accuracy and effectiveness. Their technique has many interesting features that are relevant to our current efforts to understand how to use decision-theoretic principles in control of reasoning, so we will delve into their work in great detail in §2.8 (p. 69).

But for now, let us step back and take a more general look at the issues surrounding all these different ideas about how to "correctly" propagate node values through a state graph.

2.5.3 Node value semantics

If we wish to be able to make claims about the "correctness" of a particular propagation rule for game trees, we must first define a clear semantics for node values. Having a node value semantics also gives us a standard by which to judge leaf evaluation functions; namely, we can ask how closely they approximate the intended meaning—and we can even, for some meanings of node value, use statistical models to directly implement an evaluation function that embodies the desired meaning (for example, see the models I used in my experiments in §2.5.4 (p. 47), and the model I propose in §2.11, p. 96).

For games, the simplest node value semantics is just that a node value represents the game-theoretic value of the position. We will call this interpretation *GTV-semantics.* Minimax is the correct propagation rule for GTV-semantics. If you feed game-theoretic values into minimax, you get game-theoretic values out. However, in real games evaluation functions necessarily fail to capture GTV-semantics; they instead provide some sort of crude approximation to a position's game-theoretic value. Nothing can be said about the quality of the output of minimax given this sort of input, unless we have some sort of model of the relationship between the true game-theoretic value of a position and its evaluation-function value. We saw some examples of such models, and the results about minimax that can be derived from them, in §2.4.4 (p. 37).

Another simple semantics for two-player, two-outcome games, which we will call *P(GTW)-semantics*, is that the node value represents the (reasoner's) probability that the position is a game-theoretic win for (say) Max (for multiple-outcome cases we extend this to a complete distribution over all possible game-theoretic values). If we assume that the probabilities at all leaves are independent, then the correct propagation rule for this semantics is Pearl's "probability product rule," discussed above (although in that earlier discussion, we justified the product rule using a different semantics that we will call P(W|A) semantics and will define in a moment). If the independence assumption is abandoned, then we would require a model of the dependencies to derive a correct propagation rule. No one has done such modeling as of yet, to my knowledge.

P(GTW) semantics and the product rule extend easily to games with more than two outcomes—we replace the single probabilities with distributions, and propagate then by taking products of cumulative distribution functions, as in [Baum-Smith-93].

One nice feature of P(GTW) semantics is that, in principle, we can produce evaluation functions that embody the semantics exactly, by using statistical modeling based on real data about the game-theoretic value of positions similar to the one we are evaluating. (We actually do this in the experiments reported in the next subsection.) However, it is impossible to collect data on game-theoretic values for certain classes of positions (such as most positions in chess). So, often people substitute statistics about whether a game was *actually* won from a given position in a real game. But if one does this, one is no longer obeying P(GTW) semantics; another semantics must be defined to capture what one is doing with such statistics, and we must reconsider what the correct propagation rule would be for such an evaluation function.

Another problem with P(GTW) semantics, pointed out in [Baum-Smith-93], is that it is not really the right basis for choosing a move. Just because a position has a high probability of being a game-theoretic win, that does not necessarily mean that a real player would have a high probability of winning in that position, because a real player does not have infinite computational capabilities. So P(GTW) semantics is not what we want if we wish to use node values as a basis for move choice.

Instead, suppose we say that the value of a position is our probability that, in the current game with its current real players, the Max player would proceed to win in play from that position. But let us be more precise. In particular, is this probability intended to be conditioned on the information that the position is *reached* in actual play between our typical players? Or, on the other hand, are we supposed to have no information about how we got into the position (e.g., we assume we have been plunked down into a position chosen at random)? This is an important semantic difference. To see why, consider how you would collect statistics for an evaluation function. Suppose your statistical model works by partitioning the set of board-positions into many classes, each containing many possible board-positions, and statistics are collected by counting wins and losses for each class. For the first semantics, you would collect statistics by observing many games between the two players, and for each position reached in each game, tallying the game's final outcome in the position's class. Whereas, in the second ("no information") case, you would enumerate all possible board positions, and for each position, have the players play from that position, and tally the final outcome in the class of that position. We would not expect the statistics to come out the same in the two cases, because in the first case, some of the board positions in a given class may happen to come up in real play more often than others, and the ones that happen to come up may be either more or less easily winnable for Max than the others. Thus, the two versions of the semantics are different. We will call them *P(W|A) semantics* (for Probability that max would Win in actual play from the position, given that the position is Arrived at in an actual game) and *P(W) semantics* (for Probability that max would Win in actual play from the position, conditioning on no other information).

Let us now consider what the valid propagation rules might be for P(W|A) and P(W) semantics. First, P(W|A) semantics. Suppose we have an evaluation function *f* that embodies P(W|A) semantics. Suppose the game tree looks as in figure 4, with indicated

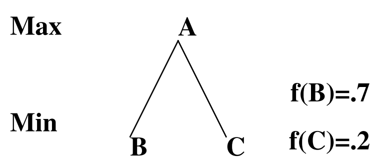
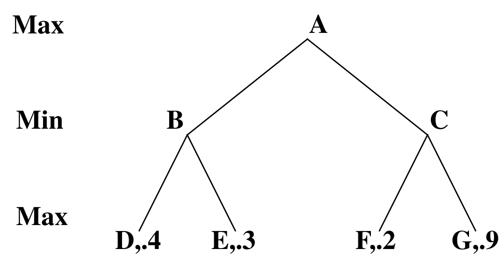


Figure 4. A very simple game tree.

values for *f*, and we are playing Max, and ourselves and our opponent are the Max and Min player(s) used in collecting the statistics for *f*, and play is currently at node A. Then clearly, we should move to B, because then the condition of arrival at B would be satisfied, and our probability of winning would be 0.7. Since we know we will definitely move to B from A, we know that our current probability of winning is also 0.7. So our propagation rule should give A a value of 0.7 in this case.

Now we consider a slightly more complicated case (figure 5). Clearly, to compute values for B and C, we will need to have some kind of model of what the opponent would do in those positions. One simplistic model would be to assume that the opponent will have exactly the same information available at position B or C as we have now. This allows us to predict exactly what min would do, (namely, go to E from B and to F from C), and the correct propagation rule for this tree would just be to use minimax (this sort of assumption was discussed in §2.4.2, p. 34).

Figure 5. A slightly more complicated game tree.



However, let us be a little more general, and just say that we have *some* kind of model of both players that yields, for each node n, a probability distribution $\vec{p}$ that gives, for each child of the node, our probability p_i that the player whose turn it is would move to that child. Then, given the vector $\vec{v}$ of the values of n's children, the propagation rule for P(W|A)-semantics is just the expectation $\vec{p} \cdot \vec{v}$.[1] This is a sort of all-encompassing propagation rule that can be instantiated in many different ways, depending on how the $\vec{p}$ distributions are computed. In practice, we will want the $\vec{p}$'s to be computed using information from node values. For example, the $\vec{p} \cdot \vec{v}$ rule reduces to minimax if we assume that all players will definitely move to the child whose value appears best for that player to us right now. (In which case each $\vec{p}$ is a unit vector pointing to the highest or lowest child.)

What if we were to assume that each player were to play in a game-theoretic optimal fashion after the current move? Well, then, our idea of a "typical, real" player had better be that of an optimum player, and so the P(W|A) semantics reduces to P(GTW|A) semantics—values are probabilities that the nodes are game-theoretic wins, given that optimal players arrive at them. This is not quite the same as the P(GTW) semantics described earlier (again, consider the difference in how evaluation function statistics

1. I first encountered this idea in [Baum-Smith-93].

would be collected—in P(GTW|A) your sample set for collecting statistics would only include nodes in principle variations, because other nodes would never be arrived at by optimal players). However, if we make a (false) independence assumption, then the expectation rule $\vec{p} \cdot \vec{v}$ will reduce to the product rule, as with P(GT).

The technique described in [Baum-Smith-93] (discussed in §2.8 of the current document), can be seen as using P(W|A) semantics, but with a more sophisticated player model that takes advantage of extra information that is propagated along with the node values. Alternatively, we can view Baum and Smith's technique as embodying a new, non-scalar semantics for node value. This semantics, which we call P(V)-semantics, is that a node value is a distribution over the P(W|A) expected-utility value that a player would assess for the node after performing a typical amount of additional tree expansion below the node, beyond what has already been done. The mean of this distribution is taken to be our current P(W|A) value for the node. Baum and Smith's technique, which we will describe later, essentially assumes that all distributions are independent, and then propagates the P(V) distributions correctly, under this assumption.

Now, let us consider P(W) semantics. P(W) semantics is not really the right way to choose moves in a real game, because we are throwing away the information that real players actually *did* arrive at the position, which should tell us *something*. It is also harder to collect statistics for a P(W) evaluation function, because we have to enumerate all possible positions, or generate them randomly according to a flat distribution (which could be nontrivial). Because of these problems, nobody ever really uses P(W) semantics; instead P(W|A) semantics is almost always intended when someone says "the probability of winning from this node." Propagation of P(W) values is just like that of P(W|A) values.

Now, the node semantics aside, we note that several times in our discussions so far, the possibility has arisen of combining probabilistic information about child nodes to derive consistent probabilistic information about parent nodes. However, we noted that to perform such a combination required some model of the dependencies between our distributions for the different children. By far the simplest kind of model of dependency is just to assume there is no dependence, and indeed, that is just the assumption that Pearl, Palay, Nau, and others have made.

But the question remains: is this assumption valid, or even close to being valid? The next section presents the results of some original investigations on this matter.

2.5.4 Experiments on sibling interdependence

As discussed above, researchers have extensively studied the product propagation rule as an alternative to minimax. We saw that the product rule involved some unrealistic assumptions about players' future knowledge of node values. But even putting those assumptions aside, the product rule involves another fundamental assumption, one that is of special interest because a similar assumption is made in an interesting piece of later research that attempts to improve on both minimax and the product rule, that is, Baum and Smith's work [Baum-Smith-93], discussed in §2.8.

In the earlier work by Pearl and others, the independence assumption was just made, without much comment. But Baum and Smith, on the other hand, explicitly stated that they expected that in real games, the amount of dependence between siblings would be small, or that in any case that any dependencies that do exist are not expected to greatly affect the quality of play. Just in case, they incorporated a tuning parameter into their algorithm that was supposed to compensate for unexpected dependencies. However, in some preliminary experiments with the Baum-Smith algorithm, the empirically-determined optimal setting of this parameter was to perform a large amount of compensation, indicating that perhaps dependencies did exist. This was a rough, preliminary result that could be interpreted in other ways, and the Baum-Smith dependence assumption is not, on the surface, exactly the same as the product rule independence assumption.

Nevertheless, in my mind, this result made me question the heretofore mostly unquestioned independence assumption used by the product rule. So I decided to determine, once and for all, if the values of sibling nodes in some real games (as opposed to the random, structureless "board-splitting" games mentioned in §2.4.4, p. 38), were interdependent. In this subsection I report on my results.

2.5.4.1 A quantitative measure of dependence

In order to make our discussion of independence more quantitative, we will wish to define, given random variables X and Y, a quantitative measure $D(X;Y)$ called the *degree of dependence* of X on Y. We will wish that $D(X;Y) = 0$ if X and Y are independent, that $D(X;Y) = 1$ if X's value is fully determined by Y's value, and that $D(X;Y)$ should have some meaningful value in between 0 and 1, otherwise. The fairly obvious answer I came up with was to define $D(X;Y)$ to be the fraction of our uncertainty about the value of X that we would expect to be eliminated upon learning the value of Y. The standard information-theoretic measure of our uncertainty about a variable X is the *entropy* $H(X)$, defined as (see [Hamming-80]):

$$H(X) = -\sum_{x} p(x) \lg p(x), \tag{7}$$

where x ranges over the possible values of X, and $p(x)$ represents our probability for the event that $X = x$.

The entropy is simply our expectation for the amount of information we would expect to gain upon learning the value of X. It is always positive, and is measured in bits, if the logarithm is taken *base 2* (as "lg" indicates it is here).

Now, we can calculate the expected reduction of entropy of X upon learning the value of Y, by considering the probabilities of the different values for Y, and the entropies of the resulting distributions $p(X|Y{=}y)$. The result is the information-theoretic measure called the *system mutual information* of X and Y, $I(X;Y)$ defined as follows:

$$I(X;Y) = H(X) - H(X|Y), \tag{8}$$

i.e.,

$$I(X;Y) = H(X) - \sum_{y} p(y) H(X|y), \tag{9}$$

which, it turns out, can be computed by

$$I(X;Y) = \sum_{x} \sum_{y} p(x, y) \lg \frac{p(x, y)}{p(x)p(y)}. \tag{10}$$

This is always be a positive quantity. See [Hamming-80] for a more detailed exposition of entropy and system mutual information.

Now, armed with the concepts of entropy and system mutual information in (7) and (10), we give a definition of dependence that exhibits the desired properties mentioned above:

$$D(X;Y) = \frac{I(X;Y)}{H(X)}. \tag{11}$$

Note that this "degree of dependence" is a very general concept that can be applied to any system of statistical variables whatsoever. However, for now we will not explore applications other than our current concern with node values.

So now, armed with this measure, we are at least prepared to explain what we mean when we say that sibling nodes are more or less independent. But now, how would we go about measuring the degree of dependence of siblings in real games?

2.5.4.2 Measuring degrees of dependence between siblings

First of all, let us state what node value semantics we will be considering. As seen earlier, the values manipulated by the product rule can be interpreted in at least two ways: the more sensible one is that they represent our probability that, with the current players, Max would win from the given node, assuming that both players immediately discover, for each leaf of the current partial graph, whether the current Max player would win from that leaf. (This does not necessarily mean knowing whether the leaf is a game-theoretic win. The discovery of the outcome of a leaf might instead involve expanding a leaf to find that all its children introduce endgames that we know from experience to be very difficult for the person playing Max, but we might not know the true value of the leaf.)

However, it is difficult to work with this sort of semantics in general, because we would need models of the players. So let us consider the other (simpler but possibly less realistic) interpretation of product-rule node values, namely that they are probabilities that the given node is a game-theoretic win. This is more satisfactory for our purposes, since we can actually determine the game-theoretic values of nodes in small games, and there is no question about how to model some vague thing like players' knowledge of each other.

To find degrees of dependence, we need a joint probability distribution over two variables, from which we can compute the value of equation (11). In our case, the two variables will be the game-theoretic value of a node, and that of its siblings. If the game is small, we can find out exactly what the ideal distribution would be, by counting, for each joint (x, y) pair, how many ways we can have a node with a value x having a sibling with value y.

But this is still incomplete. We need to specify what these "ways" are, and how they will be counted. We need to specify which nodes we are examining, and which siblings, and how many of each we are considering. The "real" joint distribution we are seeking is a meaningless concept, except when it is relative to some definite method of counting these things.

I considered several possibilities methods of counting. One was to count once for each possible node-sibling pair out of a full game-tree.[1] But then, I thought, this way of counting is biased towards nodes having lots of siblings, because the more siblings a node has, the more ways it would be counted. So, I considered normalizing the count, so that if a node had 4 siblings (apart from itself), each pair of the node and one of its siblings would count 1/4, so that we would have a total count of "1" for the node. (Actually, "2", in my program, because all the pairs would be counted in the opposite order as well, but this constant factor over all nodes makes no difference to the probability distribution obtained.)

But then, I reflected that I was being biased towards nodes deep in the tree, because there are many more deep nodes than shallow nodes. This did not seem to make sense for evaluating partial-search algorithms, because when we are playing through a game, we will spend roughly as much time early in the game doing computations on partial trees located near the top of the tree as we will spend late in the game doing computations on partial trees located near the bottom. So instead, our statistics should reflect equally nodes that might occur throughout all stages of the game.

So finally, I settled on weighting each node by its probability of occurring in a game—but that requires a model of the players again. Realistic player models were not really feasible, so as a rough approximation, I weighted each node by its probability of occurring in the game, given random moves by both players (i.e., flat distributions over the set of legal moves), or rather, the expected number of times it would occur. This can also be viewed as counting the number of times we would encounter a node in an infinite sequence of randomly-played games. (Of course, this number is infinite for all reachable nodes, but we can still define the relative counts, in the limit.)

1. Note, by exploring a *tree*, not a DAG, we mean there is no merging of transpositions. That is, our nodes correspond to histories of game-positions, rather than to game-positions themselves. In other experiments we used DAGs instead.

In the end, it turned out not to matter much—as we will see in a moment, for all these ways of counting, and for all of the games I explored, the degrees of dependence turned out to be very small; in fact, in some cases they were not significantly larger than you would get by accident in identically-structured games having random leaf assignments.

2.5.4.3 Experiments on tic-tac-toe

For my experiments I required a game whose game-graph would be sufficiently small to explore in its entirety, so that I would know for certain that the statistics were not just an artifact of the particular subset of the positions that I was exploring. Tic-tac-toe is such a game, and yet it is a game interesting enough to actually be played with interest by human players (although, admittedly, usually very young ones). For coding simplicity, I did not bother with symmetry reductions or transpositions; the game tree was still a very manageable size. Here are some basic statistics of the game tree, as computed by my program:

Table 1: Basic statistics of the tic-tac-toe game tree.

Number of nodes	549,946
Number of terminal nodes	209,088
Number of (ordered) pairs of siblings (metric 1)	740,160
Number of nodes having siblings (metric 2)	422,073
Expected number of nodes with siblings that would be encountered in a randomly-played game (metric 3)	7.273810

The last three items correspond to the three different ways I mentioned above of counting statistics for the dependency computations. For example, table 2 shows, under the second way of counting, how many of the nodes with siblings have each of the three possible game-theoretic values.

However, what we are really interested in, for each of these ways of counting, are the joint distributions between the nodes and their siblings. So, for example, table 3 shows

Table 2: Distribution of game-theoretic values for tic-tac-toe nodes having siblings. The entropy of this distribution is 1.518 bits.

	Win for "O"	**Draw**	**Win for "X"**	**TOTAL**
Number of nodes	148,708	86,101	187,264	422,073
Probability	0.352	0.204	0.444	1.0

Table 3: Joint distribution for pairs of siblings, weighted to equally represent all nodes that have siblings.

	O Wins	**Draw**	**X Wins**
O Wins	0.143	0.050	0.159
Draw	0.050	0.066	0.088
X Wins	0.159	0.088	0.197

the resulting joint probability distribution over game-theoretic values of nodes having siblings and their siblings. (Again, collected using the second way of counting.)

To illustrate what we mean by the entropy of one variable given another $H(X|\, Y)$, mentioned in equation (8), table 4 shows the distributions over node value we would get if

Table 4: Distributions of values for a node, given the value of a sibling. (Normalizing the rows of table 3.)

Sibling value	**Distribution over node value**			**Entropy**
	O Wins	**Draw**	**X Wins**	
O Wins	0.359	0.198	0.443	1.51
Draw	0.244	0.326	0.431	1.55
X Wins	0.407	0.141	0.452	1.44

a sibling value were given. Note that in the case where the sibling is a win for X, which we saw above was the most likely case, the entropy of our distribution for the node's value is decreased to 1.44 from the 1.52 entropy of the original distribution, so we are gaining some information. (If the sibling is a draw, that is unexpected, and our uncertainty about the node's value goes up; however, that case is low probability. The expected increase in information is always positive.)

The entropy of the distribution in table 2 is 1.518 bits, whereas the system mutual information of the joint distribution in table 3 is 0.0223 bits. Thus, according to our definition of dependence in (11), we get 0.0147 for the average dependence between siblings in tic-tac-toe. This does not intuitively seem to be a very high degree of dependence, and indeed, we will see that comparisons with the degree of dependence of game trees having purely random leaf outcomes bear out this perception somewhat.

But first, let us summarize the results for our different ways of counting. In table 5, X is the value of a node, and Y is the value of its sibling.

Table 5: The calculation of the degree of dependence in tic-tac-toe, for each of the three different statistical counting methods used.

Method of Counting	**Mutual Information I(X;Y)**	**Entropy H(X)**	**Degree of Dependence D(X;Y)**
Pairs of Siblings	0.04222	1.497	0.02819
Nodes with Siblings	0.02230	1.518	0.01469
Nodes w. sibs in random game	0.1875	1.503	0.12470

We can see that the degree of dependence was highest (0.12) in the case when nodes were weighted based on their probability of occurring (assuming random play), which was the counting method that we decided above was the fairest. However, that degree of dependence still seems rather low.

It occurred to me that dependencies might be highest deep in the game tree, where outcomes might be largely predetermined by moves made earlier, so I separated the statistics by depth. Table 6 summarizes the results. (Note that in tic-tac-toe it happens that all three ways I mentioned of counting statistics are identical for any given depth, because every node at a given depth has the same branching factor and the same probability of being reached in a random game. Thus we only need to give one measure of the degree of dependence here.)

Table 6: Summary of statistics for nodes at different depths. N/A means the quantity was not applicable because there were no nodes that had siblings.

Depth	# nodes	nodes w. siblings	Entropy	Mutual Information	Degree of Dependence
0	1	0	N/A	N/A	N/A
1	9	9	0	0	undefined
2	72	72	0.9183	0.0009	0.0010
3	504	504	1.4409	0.0242	0.0168
4	3,024	3,024	0.9420	0.0214	0.0227
5	15,120	15,120	1.5270	0.1358	0.0889
6	54,720	54,720	1.1887	0.0487	0.0410
7	148,176	148,176	1.5161	0.0933	0.0615
8	200,448	200,448	1.5459	0.0371	0.0240
9	127,872	0	N/A	N/A	N/A

We can see that although the degree of dependence is different at different depths, it is still never larger than 0.0889 (at depth 5).

One factor we should perhaps take into consideration, when interpreting these results, is that these joint probability distributions over the values of nodes and their siblings are conditioned on no information, other than the general method of counting nodes. But in a real program using the product rule, node win probabilities would in practice be estimated by observing some feature of the node (for example, in Nau's board-splitting research that was mentioned earlier, the feature used was the number of winning leaves below the node), and the corresponding statistics would count nodes having identical values for that feature. I considered it possible that, within some such restricted classes of nodes, there would actually be higher levels of dependence; for example, if the class picked out nodes that tended to be in regions of the tree where large contiguous regions of nodes tended to all have the same game-theoretic value.

So, I devised a simple 4-valued classifier, which classified each tic-tac-toe position in terms of whether each player had a threat (i.e., two-in-a-row with the third space empty), and collected statistics separately for each of the 4 classes of nodes. (This classifier would be a reasonable board feature to use in an evaluation function.) See table 7 for results using this classifier.

Table 7: Degrees of dependence for classified nodes. Metric 1 counts pairs of siblings, metric 2 counts nodes having siblings, and metric 3 counts nodes having siblings in a random game

		Degrees of Dependence		
Classification	**# nodes with siblings**	**Metric 1**	**Metric 2**	**Metric 3**
no threat	34,121	0.02570	0.01960	0.14140
X threat	101,968	0.05120	0.03360	0.07170
O threat	65,536	0.01640	0.02900	0.05130
X and O threat	220,448	0.00018	0.00580	0.03250
(All Classes)	422,073	0.02819	0.01469	0.12470

We can see that, even among these specific classes, the highest level of dependence is still only 0.14. Is even this level of dependence significant at all? A natural way to test this is to see is whether it is higher than we would expect to get out of a random game tree. To make the comparison as close as possible, I took the identical tic-tac-toe game tree but substituted randomly-chosen values (out of the set {X wins, draw, O wins}) at the terminal positions, propagating those instead of the real tic-tac-toe values. Results are summarized in table 6 and table 9.

Table 8: Sibling dependence for different depths in a tic-tac-toe-shaped tree with random terminal node values. Compare with table 6.

Depth	**nodes w. siblings**	**Degree of Dependence**
0	0	N/A
1	9	undefined
2	72	0.00267
3	504	0.00156
4	3,024	0.00635
5	15,120	0.00044

Table 8: Sibling dependence for different depths in a tic-tac-toe-shaped tree with random terminal node values. Compare with table 6.

Depth	nodes w. siblings	Degree of Dependence
6	54,720	0.00043
7	148,176	0.00003
8	200,448	0.00001
9	0	N/A

Table 9: Sibling dependence for different node classes and ways of counting, in a tree with random terminal node values. Compare with table 7.

Classification	# nodes with siblings	Degrees of Dependence		
		Metric 1	Metric 2	Metric 3
no threat	34,121	0.00909	0.00246	0.09967
X threat	101,968	0.00631	0.00309	0.03973
O threat	65,536	0.00271	0.00064	0.02371
X and O threat	220,448	0.00610	0.00249	0.01629
(All Classes)	422,073	0.00575	0.00232	0.07230

The first thing to note is that the degree of dependence is extremely low for nodes deep in the tree, which is what we would expect. It in interesting, however, to note that the degree of dependence increases as one nears the top of the tree. This is presumably just an artifact of the smaller sample size at the shallower depths. One disturbing thing is that we node that at depth two, the degree of dependence is actually higher than in the real tic-tac-toe game tree. This indicates that there is basically no measurable dependence at that depth in the real tree.

Let us now look at the classified-node results in table 9. We note that they are generally somewhat lower than the dependencies measured in the real tree. However, note that in metric 3, which we had decided was the fairest metric, the differences are less than a factor of two. The largest amount of dependence we have measured, e.g., 0.14 in the no-threat metric-3 case, is not much larger than the corresponding dependence, 0.10, in the random tree. This seems to indicate that most of dependency we are seeing is an artifact of our way of measuring it, and is not an important aspect of the game of tic-tac-toe.

On reflection, this is not too surprising. Tic-tac-toe does not have much of a "positional" aspect; most positions are not clearly "strong" or "weak" positions, and the number of positions is small in which one side clearly dominates the position in the sense that most of the different moves that might be made from the position had the same value. It is more often true that there are moves with many different game-theoretic values available. But we might expect that if we were to look at a more positional game where one player could get far "ahead," then at a typical node, one player would have a winning position, no matter what moves was made next; and thus a node's value will typically be correlated with who is ahead at its parent, and thus with its siblings' values, as well.

So, on the theory that perhaps the large apparent amount of independence was a special feature of tic-tac-toe, I investigated another small game played by human players that was thought to be more positional.

2.5.4.4 Experiments on Dodgem

In [Nau-82], Nau suggests an "incremental" version of his board-splitting game (called an "N-game") as an example of a game with dependencies, and shows that it eliminates minimax pathology. However, as I stated in §2.4.4, I do not consider the board-splitting model to be a good substitute for actual games; experiments on it would not satisfy me in my quest to learn about dependencies in real games.

So instead, I chose to experiment with the game of Dodgem [Berlecamp-et-al-82], which was suggested by Carl Witty[1] as an example of an incremental game that is still small enough to be amenable to a full search for purposes of collecting dependency statistics.

The rules of the game are as follows. The game is played on a 3x3 board with 2 pieces per side. The initial position is shown in figure 6. The goal for each player is to move both his pieces across the board and off the opposite side. White moves first. On his

1. In personal discussions. The version he described differs slightly from the one analyzed in [Berlecamp-et-al-82] in that a position with no legal moves is considered a draw in his version, but is a win in theirs.

turn, a player must move one of his pieces one square forward or to the side (as shown), either moving to an unoccupied square, or moving off the edge of his goal side of the board. The game ends when one player gets both of his pieces off the board, or the player to move has no legal moves. If the player to move has no legal moves, or if the game goes on forever, then the game is a draw; otherwise whichever side moves both his pieces off the board first is the winner.

In this game, the game tree is infinite (both sides can just move a piece back and forth forever), so the full tree-search method used for collecting statistics in tic-tac-toe will not work in this case. However, we note that the number of game *states* is finite and small (in fact, it is 1,963), so if we can figure out how to extract our dependency information from the game graph instead of the game tree, we can still obtain exact results.

The game-theoretic values for all positions can be computed by the algorithm shown in figure 7, which was worked out by Carl Witty and myself. The algorithm assumes that all terminal nodes have already had values out of the set {win for white, draw, win for black} assigned to them, and that all non-terminal nodes are originally marked as "draw."

The algorithm works as follows: it makes a number of passes through the graph. On each pass, it looks at each nonterminal node that is still labeled "draw." If *any* of the node's children are labeled "win" (for the player to move at the parent node), then we update the parent to "win." If *all* of the node's children are labeled "loss" (again, for the player to move at the parent), then we update the parent to "loss." If no nodes change

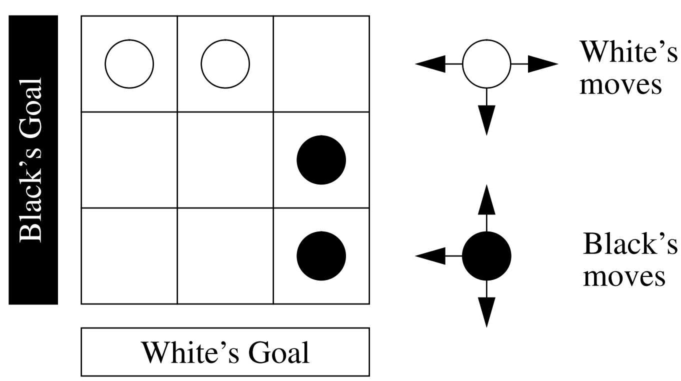


Figure 6. Initial position in Dodgem.

labels on a pass, then we are done. Why is this final labeling correct? For nodes labeled "win" and "lose," their correctness follows obviously from the definition of minimax in equations (1)-(3). For nonterminal nodes labeled "draw," the proof is still easy, but is omitted here, for brevity.

The worst-case running-time for the algorithm is $O(n^2)$ in the number of nodes n. The worst case occurs, for example, when the graph is one long chain or when the graph is fully-connected. For intermediate cases such as this game, the running time is less; in general it is $O(ndb)$, where b is the maximum branching factor and d is defined as the maximum length of any directed path not involving repeated positions that ends at a terminal node. (For this game, $b = 6$ and $d = 21$, so less than $1963 \times 6 \times 21 = 247, 338$ iterations of the inner loop are required, in fact, the actual number of iterations is only 18,574.)

Now, to collect statistics on sibling dependence, I need not only data on node values but also a counting method. (This was discussed in the previous section.) In Dodgem I cannot count pairs of siblings in the game tree, or numbers of game-tree nodes having siblings, as I did in tic-tac-toe, because in this game there are an infinite number of them. However, I can iterate over the game-positions that have at least two children, and count over pairs of their children, or over their children. It even turns out I can calculate the number of times we would expect each position to be reached in a random game, just as

```
anyChanges? = true
while anyChanges?
    anyChanges? = false
    for i = 0 to nNodes - 1
        if value[i] = draw and nSuccessors[i] > 0
            allKidsLose? = true
            for s = 0 to nSuccessors[i] - 1
                j = successor[i, s]
                if value[j] ≠ loseFor(playerToMoveAt[i])
                    allKidsLose? = false
                    if value[j] = winFor(playerToMoveAt[i])
                        value[i] = winFor(playerToMoveAt[i])
                        anyChanges? = true
            if allKidsLose? then
                value[i] = loseFor(playerToMoveAt[i])
                anyChanges? = true
```

Figure 7. Pseudocode for game-graph labeling procedure for 3-outcome games.

we did for game-tree nodes in tic-tac-toe. The resulting dependence statistics in this third case are identical to what they would be if we searched the infinite tree instead of the graph, facilitating direct comparison with the tic-tac-toe results.

The algorithm for computing these "weights" of positions, also due to Carl Witty with details worked out by him and myself, goes as follows. As before, we conduct a number of passes through the graph. On each pass, for each node, we set its weight to be the sum of weights propagated from its parents, except that for the initial position, we add 1, because that position is also reached from the "outside" (i.e., not from any predecessor position) exactly once in any game (at the start).[1] If a parent has b children, $1/b$ of its weight is propagated to each child. If the graph contains cycles (as it does in Dodgem), then some of the weight from a node will eventually cycle back to it, increasing its value still further. If the cycle has no outlet, then the weight will increase indefinitely; however, if there a path from the cycle to a terminal position (as is always the case in Dodgem), then the weight of each node will converge on a finite value (the expected value of an exponential-decay distribution). On each pass, a constant number of additional bits of the correct result will be computed, and computation will halt when all bits have been computed down to the limit of precision of our floating-point numbers. In practice, in Dodgem, only 55 passes were required for all node weights to converge to definite double-precision values.

Based on the computations, it turns out that the expected length of a random game of Dodgem is 19.358081 half-moves (moves by one side or the other), and the expected number of nodes encountered that have siblings (including the initial node) is 17.972556. When nodes having siblings are weighted according to the number of times they are expected to be encountered in a random game, the joint distribution of node and sibling values is as shown in table 10.

Next we will show the overall results for all three counting methods (pairs of children of position nodes, children of position nodes, and children of nodes weighted by their probability of occurring in a random game). Only the third measure is comparable to the data obtained for tic-tac-toe, because in the tic-tac-toe game tree the first two measures

1. In Dodgem the initial node actually happens not to have any predecessors, but in general game-graphs, the initial node may generally have predecessors.

Table 10: Joint distribution of node and sibling game-theoretic values in Dodgem, when node-counting is based on the expected number of occurrences in a random game.

Node Value	Sibling Value: Black Wins	Draw	White Wins	Total
Black Wins	0.337564	0.009809	0.141333	0.488706
Draw	0.009809	0.001341	0.019723	0.030873
White Wins	0.141333	0.019723	0.319366	0.480421

could be seen as biased towards positions that could be reached in more than one way, relative to the case in Dodgem, where the first two measures are biased against such nodes (since they only occur once in the game graph). However the third measure is measuring exactly the same thing in both cases, namely this: for a position chosen at random out of an infinite concatenation of randomly-played games, if there were alternatives to the move that arrived at that position at that point in the game, and one of those alternatives is chosen at random, what is the joint distribution over the game-theoretic value of the original move, versus the value of the alternative move? The dependence results are shown in table 11.

Table 11: For Dodgem, total counts and degrees of dependence for three counting measures. Only the third measure can be compared directly to the results for tic-tac-toe shown in table 5.

Counting Method	Total Count	Degree of Dependence
Pairs of kids of position nodes	12,464	0.284458
Kids of position nodes	5,196	0.270072
Nodes w. sibs in rand. games	17.97256	0.099197

We note that although the first two degrees of dependence are much higher than anything we have seen so far, the third dependence measure, which is considered the "fairest" one, and is the only one that can be compared directly to the corresponding result for tic-tac-toe, is, surprisingly, actually somewhat less than it was in tic-tac-toe.

Clearly, more experiments are required to understand what is going on. Some possibilities:

- Construct a game-graph version of the tic-tac-toe analyzer, and see if the degrees of dependence for the first two counting methods are higher than for the third method, as they are in Dodgem.
- We note that neither game provides us with a really large sample size, since there are not very many positions in the entire game tree. Indeed, the measured dependencies could even be partly artifacts of the small sample size, as suggested by the data in table 6 where the smaller sample sets have larger dependencies. Games where the initial position is chosen at random could offer larger data sets and more certainty that the dependencies are not artifacts.
- To begin to connect these results with efforts in more "serious" games such as Chess, Charles Isbell[1] suggests analyzing a variety of simple Chess endgames, such as the KPK class of endgames.
- To connect these results to the results in the theory of minimax pathology, we should measure dependencies of the N-type game trees studied by Nau.
- Finally, to see if these dependencies are significant, and to connect these results with the efforts to defeat pathology by using the product rule, we should analyze these games to look for evidence of pathology using both propagation rules.

The last of these items has already been done, for Dodgem; results are reported below.

2.5.4.5 Comparing minimax to the product rule in Dodgem

The degrees of dependence reported above, 0.12 for tic-tac-toe and 0.09 for Dodgem, seem small when compared to a "complete" dependence of 1.0. And yet, we saw that among only 72 nodes at depth 2 in tic-tac-toe, the degree of dependence is fifty times less—0.00267—when the leaf assignments are random. Perhaps our intuitive perceptions

1. Personal discussion.

are amiss, and these dependencies such as 0.09 are actually very large, in terms of their effect on the performance of game-playing algorithms.

To find out, I have started to perform experiments to directly measure the performance of the game-playing algorithms in the game of Dodgem. To do this, I defined a simple position-classifier, like I did for tic-tac-toe. Dodgem positions were classified according to the relative total distance of each side's pieces from the goal (13 classes ranging from white-ahead-by-6 to black-ahead-by-6). Within each class I counted win-draw-loss positions, weighted by their expected number of occurrences in a random game. This yielded a good evaluation function of positions that would map positions to the probability distribution over their outcomes (assuming they are positions out of randomly-played games). Then I used a dynamic programming method, similar to the propagation methods for game-theoretic values described above, to compute the backed-up minimax and product-rule values for all nodes and all depths. Then I collected statistics on what fraction of nodes (weighted by the expected number of occurrences in a random game) a correct move choice was made by each algorithm, as a function of depth. These data are shown in figure 8. Unfortunately, my evaluation function is too good; a correct move choice is made

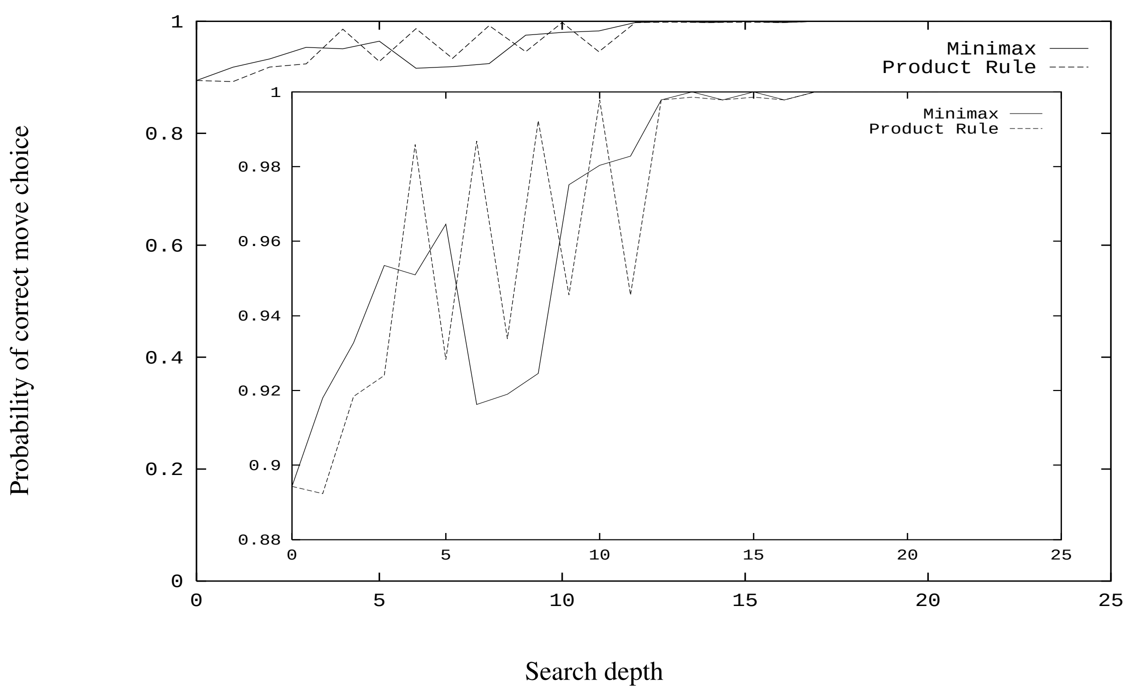


Figure 8. Probability of a correct move being made at a node for different depths of minimax and product-rule propagation. Inset shows expanded vertical scale.

at over 89% of nodes even when their children are only searched 0 levels deep (i.e., using the straight evaluation-function value of the child). Thus there was not much room for much improvement with further search. However, looking at the graph, we can see several interesting features:

- The minimax algorithm does exhibit a form of pathology, in that, for example, with a depth-6 minimax search, correct moves are made at several percent fewer nodes than with depth-5 search.[1]
- The product rule seems to be fluctuating between high and low values for many nodes. More detailed studies are needed to see exactly what is going on here.
- Both propagation rules reach 100% decision accuracy by depth 21, because in this game all nodes happen to have a solution tree of no more than that depth.
- Minimax and the product rule seem to improve, very roughly speaking, at the same rate, within the bounds of the fluctuations of the product rule curve.

Thus, we can see that perhaps the 0.09 degree of dependency in this game is, indeed, *not* sufficient to avoid pathology, and thus perhaps *is* a small value, validating our intuitive perceptions and justifying the product rule's independence assumption. However, we do not here see evidence that the product rule is doing significantly better than minimax, and the fluctuations are an annoying artifact. Perhaps further experiments on larger sample spaces will be better able to resolve the question of the importance of dependency and its bearing on the performance of the two propagation rules.

2.5.4.6 Conclusions regarding the independence assumption

What can we conclude from the above experiments? First, there are several qualifications we must make when trying to interpret these results:

- In these experiments I am measuring dependence between true game-theoretic values. However, for playing a real game with real players, we should be more concerned with the *actual winnabilities* of nodes, i.e., whether the actual cur-

1. Ordinarily one might question whether a few percent is statistically significant; however, note that in these experiments we are examining *all* nodes, not taking a random sample.

rent Max player really would win from a given node, and whether these node statuses are correlated between siblings.

- Also, the real independence issue is not just whether the actual outcomes of two nodes are independent, but also whether the evaluation-function assessments for those nodes are conditionally independent, given the outcomes.
- These statistics about nodes are collected under the assumption that the node in question is chosen randomly out of an infinite concatenation of games between random players. If we consider nodes chosen according to a different distribution, such as a flat distribution over game-tree nodes or game-graph nodes, we will get different results. None of these models of node distributions are actually *correct*; if we are concerned with the performance of an actual player in games against other players, we should really be picking nodes that would arise in the actual player's actual games. This could not really be done for the simple games we were considering, because a reasonable "actual" player would just be a perfect one, and all its moves would be to game-theoretic win nodes.
- Finally, and perhaps most importantly, I should point all that all the above dependency measures are collected between only *pairs* of nodes. But *pairwise independence does not imply mutual independence* between a set of variables. In other words, even if pairs of nodes are completely independent, there might still remain strong dependencies among sets of three or more siblings. This could be measured easily enough by taking the second variable in our joint distribution to be the vector of values of all of a node's siblings, instead of just the value of a randomly-chosen sibling. (Or, if we wish to ignore ordering information, a we could take this variable to be triple giving the number of wins, draws, and losses among the siblings.)

In summary, the results so far are very inconclusive, and more research is needed. The degree-of-dependence measure defined in §2.5.4.1 (p. 50) has not yet been firmly established as a good, solid measure that can accurately predict phenomena such as pathology, and we have not yet firmly established that real games do or do not have significantly high levels of dependence. However, these experiments at least seem to provide circumstantial evidence that perhaps the independence assumption is not too bad, and leaves open the possibility that future generations of game-playing programs may make fruitful use of probabilistic evaluation techniques that utilize this assumption.

Even if, despite these results, the independence assumption does prove to be a fatal flaw in the existing probabilistic algorithms (for some games), there does remain the possibility of developing probabilistic methods that actually *avoid* the independence assumption, via some sort of probabilistic modeling of the dependencies between pairs or sets of siblings. However, what form such a model would take remains unclear.

2.6 Selective search

So far in this chapter we have examined a variety of methods for evaluating partial state-graphs, but all under the assumption that the partial graph we were considering consisted of exactly all states that are reachable from the current state within some constant number of transitions. This is a rather arbitrary choice of partial graph. Even Shannon [Shannon-50] observed that human players generally search farther down some lines of play than others, and suggested that future versions of his computer chess player might do the same.

Early in the history of CGP, there were attempts to use "forward pruning," which avoided exploring moves that were judged heuristically to be poor choices. But these attempts were largely unsuccessful ([Smith-Nau-93a] attempt to analyze why this happened).

Later, Berliner developed a selective search procedure, B* [Berliner-79]. The B* algorithm depended on evaluation functions providing bounds to indicate a range of likely values, and propagated these bounds up the tree. This allowed search to focus on parts of the tree where tightening the bounds would help most in demonstrating that one move at the top level was better than all others. However, Baum and Smith have pointed out [Baum-Smith-93] that Berliner's attempt to discriminate one move from all the rest is not necessarily the right thing to do (if there are several equivalent best moves, B* would waste time trying to prove that one is better than the other). We will look at the question of the "right thing to do" in search control in more detail when we discuss Baum and Smith's methods in a later section.

Another general but somewhat *ad hoc* technique for selective search was developed by Rivest [Rivest-88]. His method used differentiable approximations to the min and max functions in minimax search to allow a searcher to expand the node upon which the root value depends most sensitively.

McAllester [McAllester-88] proposed a technique for selective search based on counting the number of nodes below a node whose evaluations would have to change (in "conspiracy" together) to change the node's value. Smaller number of changes are considered more likely, and thus, counting these "conspiracy numbers" allows search to be focused towards parts of the tree most likely to change the choice of best move. Some experiments with the technique are reported in [Schaeffer-91], and [Allis-et-al-91a] devises a version that uses a pruning technique similar to alpha-beta. [Allis-et-al-91] describes a specialized version of conspiracy search called proof-number search that is optimized for finding solution trees; it has been used in solving several games (proving their game-theoretic values).

All the above selective search methods are general, in that they can be applied to any number of different games. However, it turns out that special-case methods for particular games have been more popular and successful in competitive game-playing programs, so far. The method of "singular extensions," a relatively specialized technique for choosing places in the chess tree to search more deeply, was developed by the team working on the Deep Thought chess playing machine. (See [Anantharaman-et-al-90] and [Hsu-et-al-90].) Singular extensions remain the dominant technique for chess.

However, I believe that as CGP researchers focus on a wider variety of games, the more general varieties of selective search techniques will gain in popularity, since they do not have to be reinvented for each game.

But still, it is difficult to know how to compare the different selective search techniques. There does not seem to be one outstanding, obviously "right" method. And it is hard to say what the deeper principles are behind these techniques that would lead to their application outside the realm of chess-like, two-player, zero-sum games. It would be desirable to have selective search methods that taught us general lessons about control of search, and of reasoning, in a more general context. This is the motivation for the more recently developed decision-theoretic techniques which we will now describe.

2.7 Decision-theoretic search control

All of the methods for selective search discussed in the previous section seem to be trying to approximately "do the right thing" with regard to tree growth—but what exactly *is* the ideal "right thing," how can we formalize it, and how can we then implement something that more explicitly tries to approximate that formalized ideal? The tools of decision theory offer a general approach for making "right" decisions, but those tools had not even been referred to by the selective-search CGP community.

Eventually, however, these ideas began to find purchase. A method for game-tree growth based on decision-theoretic principles was described by Stuart Russell and Eric Wefald and implemented in their program MGSS*. ([Russell-Wefald-89], [Russell-90]). Russell and Wefald had a number of new fundamental insights. One was to characterize the problem of tree growth in terms of a meta-level search of possible sequences of computational actions, just as base-level game-playing characterizes the problem in terms of a search of sequences of game actions (moves).

Another insight was that the utility of a computational action can be defined in a principled way, by estimating how that computational action (such as a leaf expansion) will improve our expected utility for the overall, externally-directed decision we will make. In other words, computational utility should be tied to real-world utility. Russell and Wefald formalized this idea fairly well, at least for state-space domains with small numbers of actions. Another fundamental insight was that in order to compute computational utilities defined in this way, it was necessary to assign nodes distributions that describe what our estimate of the node's value will be in the future (after we expand it), as opposed to the distribution over game-outcomes used by product-rule tree evaluation. This constituted a fundamentally new node-value semantics, one that explicitly encompasses knowledge about not just the game-position itself, but also about our epistemology, our knowledge about the game position. This is an example of why it can be useful to be explicit about epistemology, because our programs might need to model or reason about their own knowledge in order to do well. We will delve into this matter more in chapter 3.

A third basic contribution of Russell and Wefald's work was to define the first-ever principled criterion for deciding when to stop searching: stop when the expected utility of further search becomes less than the cost of the time required to do it. (Of course, to make use of this criterion required a relatively simple model of the cost of time, one that might be appropriate for games but not for other domains.)

MGSS* contained many fundamental advancements, but it had its problems. It assumed that the error distributions at nodes were normal distributions, and this required a lot of messy approximations to properly propagate these distributions up the tree. However, at least they were trying to approximate the "right thing," instead of just devising a more *ad hoc* selective-search algorithm like singular extensions or conspiracy theory that you couldn't really say, theoretically speaking, why it would or wouldn't work. A more fundamental problem with MGSS* was the "single-step assumption," which defined the utility of a node expansion as just the utility of expanding the node and then immediately moving. This meant that MGSS* could only search until the tree reached what McAllester would call "conspiracy depth 1." Also, the asymptotic running time of their algorithm was a superlinear function of the size of the tree produced, which meant MGSS* was really not competitive with traditional (alpha-beta) search in terms of runtime. Finally, after growing the tree, MGSS* just performed an ordinary minimax evaluation of that tree, adopting the flawed assumptions of all minimax-based approaches. It seems strange to spend so much effort modeling our knowledge for purposes of tree growth only to throw away that model when considering what variation in the resulting game-tree might actually be pursued.

However, despite these drawbacks, Russell and Wefald demonstrated that their program performed much better than alpha-beta for a given number of nodes searched, in the game of Othello. But note that this does not imply that it performed better for a given amount of computation time.

MGSS* could be seen as merely a step towards more significant future developments in decision-theoretically-based game-tree search. However, Russell and Wefald's work was a milestone in CGP research in that it attempted not just to play games but to develop important concepts of meta-level reasoning and limited rationality and frame them in general terms that could be applied to areas other than game-playing as well. This attempt to be general, as I have said, is vital for any CGP research that is justified from an

AI perspective. In their pursuit of ideas applicable to general reasoning, Russell and Wefald have set a new standard to which I think all future CGP research in AI should aspire. I discuss Russell and Wefald's general limited rationality efforts in more detail in chapter 3.

Other researchers besides Russell and Wefald have worked on decision-theoretic search control for general search problems, for example Hansson and Mayer in [Hansson-Mayer-89a]. And also, others such Baum and Smith have worked on more refined methods for decision-theoretic search control in games. We will describe Baum and Smith's work in more detail in the next section, because although it does not attempt to be general, it contains many valuable ideas and tools that I believe could be generalized and combined with ideas from other work such as Russell and Wefald's, and used in other kinds of reasoning systems, such as, for example, decision-analysis systems. Some initial work to do just that sort of reapplication to decision analysis is described in the next chapter.

2.8 Baum-Smith Search

Although so far we have seen a variety of game-playing algorithms, some of which draw upon the very useful, general tool of probability theory, none of them are really very sophisticated from the point of view of someone well-acquainted with statistical decision theory.[1] Even Russell and Wefald's work, although decision-theoretically inspired, does not involve detailed, realistic statistical models, since they were more concerned with the high-level concepts they were working on. It is my view that the recent work by Baum and Smith, described here in some detail, contributes greatly to the statistical sophistication of CGP, although there is room for more exploration of the relevant statistical models.

Baum and Smith's approach to partial-tree construction and evaluation combines many of the best aspects of the earlier methods, and improves upon all of them. Their method applies decision theory to both the evaluation and construction of partial search trees, as opposed to Hansson & Mayer's BPS, which only does evaluation rationally, and MGSS*, which only does exploration rationally.

1. This is not to say that the author claims to be such himself—but he is conversant enough in that field to be able to perceive its absence in the earlier CGP work.

Baum and Smith's method evaluates partial trees by backing up probability distributions from the leaves, similarly to Russell & Wefald's MGSS*. However, the Baum-Smith method gives the distributions a different semantics that facilitates the removal of Russell & Wefald's single-step assumption from the tree-growth procedure. Moreover, Baum and Smith's method doesn't require normal distributions, and so they are able to back up their distributions with no loss of information. They represent their distributions in a form that allows for extremely simple exact propagation up the tree.

Baum and Smith produce their partial trees by incremental expansion from the root state for as long as time permits. Their trees are grown by greedily expanding those leaves having the highest expected utility, as in MGSS*. However, Baum and Smith abandon Russell & Wefald's single-step assumption in their definition of the utility of leaf expansion, thus allowing the tree growth procedure to expand the tree beyond conspiracy depth one.[1] Using several mathematical and computational tricks, they approximate their ideal tree-growth procedure in an efficient algorithm.

The next subsection describes Baum-Smith tree evaluation, while §2.8.2 describes tree growth.

2.8.1 Evaluating partial trees using BPIP-DFISA

Baum and Smith's method for extracting information from partial trees is called BPIP-DFISA,[2] for "Best Play for Imperfect Players, by Depth-Free Independent Staircase Approximation." BPIP-DFISA, like most partial-tree processors, works by *evaluating* the tree, deriving *values* (expected utilities) for nodes from the values of their children; values of leaves are obtained from examining only the leaf itself. However, Baum and Smith deem the ancient minimax definition of the value of interior game-tree nodes to be unsatisfactory, due to its assumptions of perfect play by both players and of the correctness of leaf values. They propose an alternative definition which is based on probabilistic modelling of the future value assessments of the agents involved, and describe a technique for doing this modelling.

1. Meaning, we can continue expanding even if no single node expansion can possibly change our choice of best move. MGSS* was not able to do this.

2. Best Play for Imperfect Players, by Depth-Free Independent Staircase Approximation.

My description of BPIP-DFISA is broken down into several parts. First, the basis for the technique is a simple conceptualization of the problem called "the ensemble view" (§2.8.1.1). From this is derived a recursive strategy for node valuation called "Best Play for Imperfect Players," or BPIP (§2.8.1.2). However, BPIP requires a probabilistic model of future information in order to be concrete. For this, the authors provide the method called DFISA, for "Depth-First Independent Staircase Approximation" (§2.8.1.3). DFISA, in turn, requires an evaluation function that returns good probability distributions; Baum and Smith describe a statistics-gathering technique for generating such (§2.8.1.4). Finally, information gathered at the leaves must be propagated up the tree; a few simple operations are described for doing this (§2.8.1.5).

2.8.1.1 The ensemble view

Baum and Smith's simple conception of partial-tree valuation is as follows. We have a partial tree T. There is presumed to be some unknown *assignment* A of true game-theoretic values to all the leaf nodes in T. We imagine that for each leaf node λ of T we have a probability distribution $\Phi_\lambda(x)$ over the value that A assigns to λ.

Assuming that all the Φ_λ's are independent, we can derive from them a distribution over the possible assignments a, as follows:

$$\Phi(a) \;=\; \mathrm{Prob}\,\{A{=}a\} \;=\; \prod_{\lambda} \Phi_\lambda(a(\lambda)), \tag{12}$$

where λ ranges over all the leaves of T, and $a(\lambda)$ is the value that a assigns to λ. This distribution is called the *ensemble*, and it allows us to define a distribution over possible values for each node ν in T, as follows:

$$\Phi_\nu(x) \;=\; \sum_{a:\ (\upsilon_a(\nu)\,=\,x)} \Phi(a), \tag{13}$$

where $\upsilon_a(\nu)$ is defined as the game-theoretic value of ν, given the leaf assignment a. This formula says that our probability of a node ν having the value υ is just the sum of the probabilities of all leaf assignments in which ν has that value.

The problem with this ensemble method of defining node value is that it's pretty far removed from anything approaching an actual algorithm. Summing over all possible trees is not something we can do in practice; we need a simple recursive definition of node value that we can efficiently apply. This is the role of the BPIP definition; it is equivalent to the ensemble view but is expressed in terms that don't require large constructions like the ensemble.

2.8.1.2 Best Play for Imperfect Players

BPIP is a node-evaluation strategy by which an agent or *player* in a 1-agent (or 2-agent zero-sum) strategic situation[1] can determine the child of the root that maximizes his expectation of the value[2] of the final outcome of the interaction, given only a partial *game-tree* of reachable states. The BPIP strategy assumes, as minimax does, that both players are using the strategy and that this is common knowledge. However, BPIP abandons minimax's incorrect assumption that the leaves of the partial tree are terminal states and that their static values are exact and correct.

The following definition of BPIP is a formalization of the definition that Baum & Smith informally describe.

First, some notational conventions:

- T means a partial search tree.
- ν_T indicates the root node of T.
- T_ν means a partial search tree that is rooted at the node ν.
- α is an *agent.*
- α_T is the agent α in a state of knowledge where it has direct knowledge of only those nodes that are in T.
- We write α^ν for the agent whose turn it is to move at node ν.[3]
- Υ_ν is a random variable for the desirability of the final outcome of game-play if the current players were to play from node ν.

1. There may be chance events, but actions by the two agents and by nature must be non-simultaneous, and instantly known to both players. Work remains to be done on how to generalize BPIP to strategic interactions with simultaneous play or with more than 2 agents, as has been done for minimax (see [Bodkin-92]).

2. Values can be thought of as expected utilities. I use the term "value" instead of "expected utility" in this discussion because Baum and Smith do so, and it is shorter.

3. We assume that there is a definite agent whose turn it is at each node. For chance nodes, "nature" is treated as an agent. This was done in [vonNeumann-Morganstern-44].

- $\wp_\alpha\{P\}$ is α's subjective probability that the proposition P is true.
- $E_\alpha[X]$ means agent α's expectation for the value of the random variable X:

$$E_\alpha[X] = \sum_x x \wp_\alpha\{X{=}x\} . \tag{14}$$

- An agent α_T*'s subjective value* $\upsilon_{\alpha_T}(\nu)$ is defined over nodes ν in T as:

$$\upsilon_{\alpha_T}(\nu) = E_{\alpha_T}[\Upsilon_\nu] , \tag{15}$$

  in other words, the expected payoffs in the final outcome of play between the current players from ν (at a time when the players will have access to a tree rooted at ν).

- Finally, α_T's *best play* $\beta_{\alpha_T}(\nu)$ from a node ν in T is that child ν_i of ν having the best subjective value $\upsilon_{\alpha_T}(\nu_i)$, from α_T's point of view. α_T may be minimizing, maximizing, or choosing randomly (e.g., if α is "nature").

The definition of the BPIP strategy is a consistency constraint on an agent's subjective values for nodes. Agents must be obey this constraint to qualify as obeying the BPIP strategy. This constraint is expressed in the form of a recursive definition of the values of a non-leaf tree node in terms of the values of its children.

Definition. Best play for imperfect players (BPIP):

An agent α is obeying the BPIP strategy if and only if, when given only a partial tree T of nodes to access, α's subjective value function υ_α obeys the following recurrence for each interior (non-leaf) node ν in T:

$$\upsilon_\alpha(\nu) = E_\alpha[\upsilon_\alpha(\beta_{\underline{\alpha}}(\nu))] = \sum_i \upsilon_\alpha(\nu_i) \wp_\alpha(\beta_{\underline{\alpha}}(\nu){=}\nu_i) \tag{16}$$

where $\underline{\alpha} = \alpha^\nu_{T_\nu}$, that is, the agent α^ν whose turn it will be at node ν, at a time when the progress of the game has actually reached node ν, and α^ν is in a knowledge state wherein he can access a different partial tree T_ν of nodes below ν.[1]

1. α need not know exactly what this tree will be since it may extend beyond T, the limit of α's knowledge. α's lack of knowledge of the identity of T_ν is encompassed in α's overall uncertainty about the value of $\beta_{\underline{\alpha}}$.

Additionally, if $\nu = \nu_T$ (the root of T) so that this time is now and $\alpha^\nu = \alpha$, BPIP requires α to realize that T_ν is just the current tree T, and $\underline{\alpha}$ is just α. Then (16) simplifies to:

$$\upsilon_\alpha(\nu) = \upsilon_\alpha(\beta_\alpha(\nu)). \tag{17}$$

To sum up (16), BPIP requires that an agent define its subjective value for interior non-root nodes to be its expectation of the value of the move that would be made from that node. Equation (17) says that the value of the root is the value of its best child.

Now, the key difference between BPIP and minimax is that BPIP does not require explicit maximization except at the root node; the value of an interior node is instead the sum of the children's values, weighted by the probability that the child will be chosen by α^ν. Thus BPIP handles future actions by both players as uncertain, chance events. This is the right perspective, given the fact that, even if we know exactly how the player α^ν works, we do not know exactly what information will be available to him then that isn't available to us now, and thus, his choice of move is uncertain.

Now, BPIP is an incomplete strategy in the same way that minimax is. Like minimax, it neglects to specify how an agent gets its leaf-node values υ. However, unlike minimax, BPIP also depends on personal probability distributions over the $\beta_{\underline{\alpha}}(\nu)$ variables indicating which moves will be made at future decision points. Minimax assumes that all future decisions will be made just the way that we would make them now.

So, a BPIP agent is assumed, unlike one using minimax, to have some degree of knowledge about the agents involved in the game, including itself; knowledge that lets it assess how these agents will play in future situations in which different information is available. This knowledge is expressed not only by the distributions over β's but also through the υ's of the leaf nodes. BPIP, unlike minimax, imposes a particular interpretation on leaf node values, namely, that they represent an expected value for the desirability of the final game outcome, given that the current players are continuing to play from a given node but with more information about the tree below that node.

The authors note,

> In order to make BPIP concrete, one needs a *probabilistic model of extra information* so that one can estimate the probabilities in [(16)]. If one chooses a

> sufficiently tractable such model, then it becomes possible to actually compute BPIP ([1], p. 7).

They go on to describe their chosen probabilistic model of extra information, which they call "Depth-free independent staircase approximation (DFISA)." This is described in the next section.

2.8.1.3 Depth-free independent staircase approximation

We have seen that BPIP does not constrain the values of leaf nodes. Of course, to instantiate BPIP we must come up with a way to derive leaf values, just as in other partial-tree-evaluation paradigms. However, in BPIP we must provide something extra as well.

Recall that (16) required, for each interior node, a probability distribution over the moves that a player might make at that node. A BPIP agent must provide this distribution. Can we derive this distribution solely from the values of the child nodes? If one were willing to assume that the player's future assessment of the value of the children would be the same as our current value for them, then yes, we could just declare the child that gives the best value to that player to be the one that is certain to be chosen. This is ordinary minimax. However, the point of BPIP is to move beyond this assumption, and model the fact that the valuation of a node will generally have changed by the time that node is reached in a real game. How can we model this possibility of change?

Baum and Smith's answer is to model it probabilistically. They broaden the notion of node evaluation to include not just the expected value of a node, but also a distribution over the possible future expected values for the node, that is, the values we might get for it after we do "more expansion" below it. In DFISA, the precise meaning of "more expansion" is intentionally left undefined; DFISA makes the fundamental assumption (the "depth-free approximation") that *any* amount of further expansion would have approximately the same effect. The idea is that the important difference is the difference between some expansion and no expansion at all. In practice, "more expansion" has a fairly definite meaning, namely, the expansion that would be done by Baum and Smith's tree-growth algorithm in a typical amount of remaining time. This is still not precise, but it is assumed by Baum and Smith to be good enough.

In any case, armed with these node-value distributions, let's look again at the problem of finding the probability that a player would make a certain move in some situation. If we knew what new values the player will have assigned to the node's children by the time the node is reached, we could predict which child the player would pick as his move. We already have distributions over the values that will be assigned to individual children, and so if we just assume independence (the "I" in DFISA), this allows us to get distributions over entire assignments of values to all the children—like in the ensemble view of §2.8.1.1. (Note that here we are talking about distributions over future subjective expected values, rather than distributions over true game-theoretic values, as in the ensemble view. However, the means of these two kinds of distributions are assumed to be equivalent.) Given these distributions over assignments, we can predict, in theory, the probability that a certain move will be made by summing over the assignments in which that move is the best one.

In practice this summing would be infeasible, and Baum and Smith's actual proposal for the implementation of DFISA avoids this work, and without approximation, by taking advantage of a special "staircase" (the "S" in DFISA) representation for the distributions. This will be discussed more in §2.8.1.5.

2.8.1.4 Leaf evaluation

BPIP-DFISA requires a value for each leaf, which is the mean of a probability distribution over the values that would be obtained by future search at that leaf. Thus, we need an *evaluation function* that, given a leaf, can provide a reasonable value and distribution. Baum and Smith propose a specific method for doing this for games like chess. It would work by collecting statistics from a large database of games and their outcomes. They describe the method briefly, but do not go into detail; it is not presented as a main contribution of their work.

The leaf evaluation task is broken into two functions: a function E_1 that produces an expected value for the node, and another function that returns an error distribution relative to this value. In Baum and Smith's proposal, functions for the two parts are learned separately, from different data. E_1 is just like an ordinary evaluation function, and can be learned in the normal way, but learning the error distributions requires a separate, special method.

To produce E_1, they manually define a classification tree for chess positions, that classifies any position into a class or "bin." For each position in each game in a large library of well-played games, they categorize the position into one of these bins. Within each bin, they fit a multilinear function of positional features $x_1, \ldots, x_n$ to the utility of the game's outcome. Thus, to predict the value of a new position, they simply look up the position's bin, find its feature vector, and run it through the linear function learned for that bin.

To produce the distribution-returning function, they produce another (possibly different) classification of positions into bins. They play a large number of games using E_1 with ordinary minimax search, and for positions searched in these games, they record the change in the node's E_1 value obtained by deeper search (for example, depth 6). Recall that they assume the exact amount of deeper search not to be very important. They compress the set of data-points in each bin into a simpler representation of the distribution of points; the exact method for this is described in a separate paper [Smith-92].

2.8.1.5 Backing up distributions

Baum and Smith's method for quickly backing up distributions depends heavily on the particular "staircase" (the "S" in DFISA) representation that they use.

The ideal of a continuous probability density function is approximated instead by a distribution consisting of a sum of delta functions, i.e., a set of spikes. This can be thought of in mechanical terms as a set of point probability masses p_i at different positions x_i, where "position" means the value in the domain of the distribution.

Second, this representation can simultaneously represent the "jumps" in a pair of staircase *cumulative distribution functions* that give, for each value x in the domain, the amount of probability mass located above (below) x. Formally, the upper and lower CDF's $\bar{c}^{(\nu)}$ and $\underline{c}^{(\nu)}$ of a node ν are defined by:

$$\bar{c}^{(\nu)}(x) = \sum_{i : x_i^{(\nu)} \geq x} p_i^{(\nu)} \tag{18}$$

and

$$\underline{c}^{(\nu)}(x) = \sum_{i: x_i^{(\nu)} \le x} p_i^{(\nu)} \tag{19}$$

where $p_i^{(\nu)}$ and $x_i^{(\nu)}$ are the probability masses and abscissae (positions) of our spikes i for the distribution of ν.

The advantage of the CDF functions is that they permit the exact computation of a distribution for a node ν in terms of the distributions for its children ν_i, in a way that is consistent with BPIP but skips the step in (16) of computing probabilities of moves.

The formula for the lower CDF of a node ν in which the agent to move at that node is attempting to maximize his value is:

$$\underline{c}^{(\nu)}(x) = \prod_i \underline{c}^{(\nu_i)}(x). \tag{20}$$

The formula for the upper CDF is similar. If we wish to have a minimax (as opposed to minimax) representation of value, we can substitute $\bar{c}$ for $\underline{c}$ and $-x$ for x on the right side of (20).

If the player at ν is "nature," then the formula is instead:

$$\underline{c}^{(\nu)}(x) = \sum_i \wp_i^{(\nu)} \underline{c}^{(\nu_i)}(x), \tag{21}$$

where the $\wp_i(\nu)$ are current agent's assessments of the probabilities of the different children given the parent.

Baum and Smith say that (20) can be computed quickly and exactly by something like a binary list merge on the set-of-spikes representations of the children, taking only $n \log n$ time in the number of spikes. With this technique, no information is lost as distributions are propagated up the tree.

2.8.2 Constructing partial trees

Baum and Smith apply the decision-theoretic approach to the construction as well as the evaluation of partial search trees. They grow their partial trees incrementally, expanding the tree a little at a time, with the choice of what part of the tree to expand being guided by information obtained during the evaluation of the result of the previous tree.

However, BPIP-DFISA tree evaluation takes time proportional to the number of leaves in the tree, so if the tree were deepened one leaf at a time in this way, the running time would be quadratic in the size of the final partial tree produced. Baum and Smith avoid this problem using the "Gulp trick," in which a constant fraction f of the leaves in the tree are expanded before each reevaluation. This ensures that the running time is dominated by the time taken by the final tree evaluation, like in traditional iterative deepening. The gulping trick is described in more detail in §2.8.2.1.

So with gulping, the goal in each stage of tree expansion is to expand that set of k leaves whose expansion will have the highest expected utility. In order to make this feasible, Baum and Smith do two things: (1) they assume that there is an additive measure of leaf expansion importance such that the value of expanding a set of leaves is the sum of the importances of the leaves in the set. Although they provide a counter-example that disproves this assumption, they hope it will work well enough in practice. (2) They assume that expanding a leaf always makes its distribution collapse to a single point, so that expanding the most important leaves amounts to expanding those leaves whose true values we would most like to know. Baum and Smith note that making wrong assumptions such as these in tree construction is less critical than making them in tree evaluation, since they are only being used to guide tree growth and not to choose actual moves; they affect efficiency of search rather than quality of play. Also, even with these simplifications, Baum and Smith expect their search to dominate both flat search and MGSS* search.

The measure of leaf expansion importance that they define is different from Russell & Wefald's. Recall that MGSS* assumed that the utility of expanding a leaf was equal to the utility of expanding it and then immediately moving: the "single-step assumption." The single-step assumption is bad, because it means that when the tree's conspiracy depth becomes greater than 1 (i.e., no single leaf expansion can change our choice of best move), which happens very quickly in practice, MGSS* will conclude that no expansion has utility higher than the utility of just moving without further search, so it will terminate tree-growth.

Baum and Smith instead define leaf expansion importance with a measure they call "Expected Step Size," or ESS. The ESS of a leaf-expansion is essentially the absolute amount that we expect it will change our expected utility for all possible future expansion.

In other words, if we think that expanding a certain leaf will clarify, one way or the other, the value of future expansion, then that leaf is a good one to expand. If it lowers our utility for future expansion, it prevents us from wasting time doing more expansion, and if it raises our utility for expansion, it prevents us from quitting and making a move too early. Baum and Smith's quitting criteria are discussed in §2.8.2.5.

Interestingly, ESS can be thought of as an instance of a more general principle of limited rationality, which is that learning more about the value of actions is a very valuable action in itself. When you are starting to do a round of expansion, first consider what you will do afterwards. There are two possible actions: you can either move right away after this round, or you can continue doing more expansion. You have some current utility for each of these actions, and some current best action. However, these utilities are uncertain. You might have a distribution over how your utility for each of these actions will change after this round of expansion. If these distributions overlap, then you would like for your next round of expansion to tighten up those distributions, so as to tell you for certain whether or not to go on and do another round.

Baum and Smith don't have a distribution over the utility of continuing to compute, or a model of how much certain leaf expansions will tighten it. Instead, they estimate how much a particular leaf expansion will perturb the mean of the distribution; the assumption being that large perturbations correspond large tightenings of the distribution.

ESS can be computed using the concept of "influence functions," which express the amount that some quantity at the root node will change if some quantity at one root changes while other roots remain fixed. These will be described in §2.8.2.4.

2.8.2.1 Gulp trick

The "gulp trick" is a cost-amortization technique that enables Baum and Smith's tree-growth algorithm to run efficiently despite the requirements of ESS-controlled tree growth. The trick, as mentioned above, is to expand not just the single most important leaf at each step, but instead the $\lceil nf \rceil$ most important leaves, where n is the current number of leaves and $0 < f \leq 1$ is a constant fraction determined empirically. Thus the cost of reevaluating the tree after each growth step is amortized over a large number of new nodes produced, and the total cost of all steps is dominated by the time for the final tree evaluation.

Baum and Smith describe the efficiency of gulping in terms of a constant *slow-down factor* C which describes, given f and the average branching factor b, how much time the tree growth (with gulping) takes compared to the time to evaluate the final tree produced—assuming that the average branching factor is larger than 1. The slowdown factor for chess is computed to be 2.2 for $f \cong 0.02$.

Now, the gulp trick is an approximation; more ideal would be to reevaluate the tree after each node expansion. In some circumstances, gulping might grow much worse trees than the ideal. To illustrate, imagine a chess position in which there is a single, long, clearly-forced sequence of moves following that position, where the ideal would be to trace down that forced sequence to see how it comes out, before examining other possibilities. With gulping, however, we would expand other nodes in parallel with each step down the forced path; a larger and larger set of them each time. So we would not get very far down the forced path.

Still, we expect to get farther down the forced path than flat alpha-beta would. And we don't usually expect there to be only one forced line like this; in general we expect that the number of reasonable variations increases exponentially with tree size; if $f+1$ is tailored to fit this exponent, then gulping should usually do well. We cannot make f too small, however, because then the slowdown factor will increase.

In summary, gulping seems like it ought to do fairly well for chess. To handle a more general range of problems, we might want the tree-growth algorithm to try to adapt the value of f to fit the needs of the problem at hand, but Baum and Smith don't discuss this.

2.8.2.2 Leaf expansion importance

Given our "gulping" policy of expanding a certain number k of the leaves of the current tree before proceeding, the problem within that policy is how to find the set of k leaves whose expansion would give us the highest expected utility. Baum and Smith describe a solution that gives the ideal decision-theoretic way to derive the utility of expanding any subset of leaves, given the CDF's for the children of the root, and the assumption mentioned earlier about distributions collapsing to a point. The definition essentially sums over all possible moves that might be found to be better than our current best move, and over the amounts that it might be better, if that set of leaves were expanded. (This is similar to one of Russell & Wefald's definitions.) However, to apply this definition to the problem of choosing the right subset of leaves, it seems necessary to apply it to all the subsets, which is clearly infeasible.

So instead, Baum and Smith simplify the problem by assuming that the utility of expanding any set of leaves is just the sum of a measure, called the *leaf expansion importance,* that is evaluated for each leaf in the set. They in fact prove, with a counter-example, that no such measure can exist. However, they proceed in the hope that it will "often be approximately true," as with their other simplifying assumptions.

One possible importance measure, which Baum and Smith call "Single-Leaf Expansion Utility" (SLEU) would incorporate Russell & Wefald's single-step assumption. However, Baum and Smith dismiss this measure for several reasons, including its limitation to conspiracy-depth-1 trees.

Instead, the authors do the following: they evaluate the above-mentioned true utility of expansion for the set of all leaves, and consider this to be the *expected utility of all future expansion (EUAFE).* In other words, it would be the expected utility of knowing the true values of all the leaves in the tree. To make sense of this, think of it as the expected cost of proceeding with our current guesses for the leaf values instead—in other words, how likely is it that our current guesses are wrong.

This EUAFE measure is then used to define the ESS leaf-importance measure, in a way that implicitly seems to identify the EUAFE with the expected utility U_{continue} of the action "continue to expand the tree" (as opposed to "stop and make a move now"), although Baum and Smith do not themselves make this identification. Since, in continuing to expand the tree, we will probably not actually expand all the leaves, this interpretation of EUAFE is not exactly correct.

2.8.2.3 Expected Step Size

The "Expected step size," or ESS, of a leaf is defined to be the expected absolute value of the change in the EUAFE that would be caused by expanding that leaf (reducing its distribution to a point) while keeping other leaves' distributions the same. As I described in the introduction to §2.8.2, this can be seen as an approximate measure of the expected improvement in the quality of our decision about whether to continue doing more expansion after the current round of it is over.[1] Unfortunately, Baum and Smith do not attempt to justify ESS in these terms, and instead put forth a number of less satisfying arguments of why ESS is a good measure. They do, however, provide an approximation theorem that describes ESS in utilitarian terms.

One nice thing about the ESS measure is that it can be computed quickly from a node's influence function. An influence function describes how changes in the node will affect the root's value. The influence function can, in turn, be computed in time only linear in the number of nodes in the tree. Baum and Smith provide fairly detailed pseudo-code for the computation of ESS and influence functions in an appendix, so I will not attempt to reinvent their algorithm here.

2.8.2.4 Influence functions

In Baum and Smith's algorithm, influence functions are computed top down, with each node's influence function computed from its parent's. This algorithm, like the BPIP-DFISA algorithm, depends strongly on the representation of distributions as a set of spikes. This representation allows a node's influence function to be represented as a vector giving the partial derivative of the root value with respect to the height of each spike in the node's distribution. Baum and Smith observe that this partial derivative is just a constant, because their propagation functions ((20) and (21)) are just linear in the CDF's of root nodes. These constants can be computed for a child based on the distributions in its siblings and parent and on the parent's influence function.

Thus, Baum and Smith's influence-function computation works by traversing the tree downward from the root, accumulating factors into these constants on the way down the tree. The total work is proportional to the number of nodes.

2.8.2.5 T ermination of growth

Finally, Baum and Smith suggest a natural termination condition for search in games, which is to consider that there is some additive utility cost for CPU time, and,

1. As mentioned in §2.8.2.2, I interpret EUAFS as an attempt to approximate U_{continue}, the utility of continuing to search instead of stopping after this round of expansion.

assuming we can estimate the time required to do the next round of expansion, searching should cease when the EUAFE becomes less than the current utility minus the cost of that time, similarly to how Russell and Wefald terminate search (note that this assumes that we will get most of the benefit from future expansion in the very next round of expansion). For a timed game like chess, Baum and Smith suggest time-cost could be proportional to the estimated number of moves left in the game divided by the time remaining, using a constant factor determined by experiment.

However, this way of handling time is not very theoretically satisfying. There is very little that can be said about exactly why, and when, and how well this technique will really approximate maximization of our probability that we will win the game, or find the solution on time. What will happen if the rules of the game depend on time in a more complicated way than in chess? Even for chess, this treatment of time is weak: for example, suppose the opponent is running low on time and we are not. To maximize the probability of winning, we ought to try to steer play into complicated situations, where careful thinking is required in order to avoid making a blunder, in the hope that the opponent, in his rush, will make a mistake. Treating time as external to the domain, and only using it to cap computation time, does not address this kind of reasoning at all. I will discuss the issue of time some more in a later section.

2.8.3 Problems

Baum-Smith search has not yet, to my knowledge, been implemented by anyone, for any domain, so its methods have yet to withstand the rigors of empirical study. The authors are working on implementing it for the domain of chess, using a handmade classification space and a large library of well-played chess games to train up an initial evaluation function.

In doing away with some of the assumptions of traditional minimax, Baum and Smith have to make quite a few new, untested simplifying assumptions and approximations. Many of these are fairly well-justified by Baum and Smith's theoretical arguments, and it seems that the problems are not too serious, but it remains to be seen how well they will work in practice. A list of some of the assumptions and approximations:

- Depth-free approximation (§2.8.1.3).
- Gulp trick (§2.8.2.1).
- Additive leaf expansion importance (§2.8.2.2).
- Additive time cost (§2.8.2.5).

Additionally, Baum and Smith's method is currently not very approachable; parts of their description are very difficult to understand. However, it is a draft paper, and I expect that their presentation of their ideas will improve with time. I hope that this thesis will, as a supplement to the Baum and Smith draft paper, serve to clarify some of their ideas.

2.9 What' s Next for CGP?

Baum and Smith close their paper with the following insightful comment on the lack of generality of the entire partial-tree-search paradigm for game-playing:

> We conclude with the remark that this still has little to do with how humans play games. The computer science approach (which we are attempting to perfect) has since Shannon basically regarded a game as defined by its game tree. But what makes a game interesting is that is has a low complexity, algorithmically efficient definition apart from the game tree.... Any procedure which only accesses the underlying simplicity of a game in the form of an evaluation function is inherently doing the wrong thing.... The main open question is how to go beyond the evaluation function picture of games.

I heartily agree with this sentiment, and I would like to propose that it applies not just to game-playing but to the similar search methods often used in puzzle- and problem-solving domains. I also propose that in questioning the current approach, we ought to try to reveal all of its hidden assumptions (which I attempted to enumerate earlier), and determine which are most necessary for future progress, and which would be beneficial to try working without. Sections §2.9.1 and §2.9.2 reexamine a few of these assumptions and suggest how they might be relaxed. Sections §2.9.3 and §2.9.4 examine some unrelated methodological issues.

2.9.1 Generalizing the notion of games

In the typical notion of game-playing dealt with in AI, there are many implicit assumptions that remain invisible until pointed out. Games are assumed to have only two players. Moves are made in a definite sequence, rather than simultaneously. There is always a definite player whose turn it is to move. Work by Ronald Bodkin has shown how to loosen these assumptions and generalize some of the traditional game-playing algorithms [Bodkin-92]. Can this be done for Baum and Smith's approach as well? It seems likely that work can be done along these lines; generalizing scalar outcome-values to payoff vectors, generalizing move lists to multi-player move matrices, etc. This is one way that the Baum and Smith approach might readily be generalized.

2.9.2 Abstract search

A fundamental issue in treating any domain as a state space is the choice of which properties of the world will be included in your notion of a state (or concrete microworld state, in the terminology of section §2.3). The normal game-tree notion of state includes things relevant to the rules, like where the pieces are, and whose turn it is. However, normal game-tree states often miss a couple of things that may well be relevant: the history of moves, and the current time. In chess, the history of moves is important because of the rule forbidding thrice-repeated sequences of positions, and the time is important because you lose if you take too much time on your moves. Ignoring these properties leads to problems in chess, and proposed solutions (such as in Baum and Smith's appendix G) usually have an unsatisfying, *ad hoc* flavor.

So why don't people include these properties, which are, after all, relevant to the game, in the concept of a game state? The reason is that CGP researchers realize that including these properties would ruin the performance of their game-search algorithms. Why? Because all of a sudden there would be many more states than before, many of them differing only in these two properties. It multiplies the size of the problem a thousandfold.

But since these properties are important to they game, how can game-programs play well while ignoring them? After all, games researchers don't ignore other properties, such as the positions of pieces.

The answer is that the games researchers have encoded, in their definition of state, domain-specific knowledge to the effect that these properties *usually aren't very important* to things we care about, such as the expected utility of a state. Usually, in a game like chess, the utility of a board position is the same no matter how you got there, or exactly what time it is. In other words, the program designers have built in an abstract notion of state, not a concrete one at all, because they realize that the abstraction is useful; it provides a *good approximation* to the more computation-intensive ideal of considering the position history and game time.

This raises a number of very interesting questions:

- If we can abstract over these properties, would it perhaps be useful to consider abstracting over others? For instance, wouldn't it be nice to realize that my planned fork will work no matter where my king is, so that I don't have to invent it again when considering moving my king? In solitaire, wouldn't it be nice to be able to think about moves I can do after revealing some hidden card, regardless of the identity of that new card? (We will discuss this case in more detail later.)
- Could the kind of abstraction done be different depending on the situation being reasoned about? After all, sometimes time and history *are* important. Wouldn't it be nice to do the right thing in such cases, without requiring a separate ad-hoc mechanism for that?
- The existing kind of abstraction just assumes complete equivalence between states having only the abstracted-over feature(s) different between them. Without any abstraction, we traditionally assume all states are distinct. Is there perhaps a middle ground, where you take the value of one complete state as probabilistic evidence for a similar state's having a similar value? And have the degree of strength of these relationships be based on real, statistical data about the correlation?
- How do you define the value of an abstract state, not knowing how that state will be instantiated? (My proposed answer: use the *expected* value if it were to be instantiated.)

- Perhaps this issue of abstraction is the reason for Baum and Smith's observation that humans play games very differently—perhaps humans are abstracting over states in many ways, all the time, rather than always considering complete states.
- Could we perhaps, in moving to this idea of abstract states, bring with us some of the useful ideas from concrete-state-space search? E.g.: state graphs, partial search, incremental refinement, propagation of values through the graph, distributions over future expected values, rational evaluation, rational control of search.

Most of these questions remain unanswered, and even rather poorly-defined. At first they only seem to hint at a future direction. However, in the research discussed in chapter 4, I have begun to make some progress towards clarifying and answering some of these important questions.

One preliminary conclusion is that transplanting rational techniques from concrete state-space search to abstract state-space search is probably not so hard after all. It is made easier by the observation above that existing game-state-spaces are really abstract spaces already, and thinking of them in those terms makes it easier to see how the current methods might make sense with other kinds of abstraction as well.

One might think that it would be easier to first try transplanting simple techniques like minimax before attempting to generalize all of Baum-Smith search to a world of abstract states. On the contrary, I think that a decision-theoretic approach is vital for properly handling all of the uncertainties and all of the questions of usefulness that are involved in any sort of abstract reasoning. I think that some of the previous attempts in AI to do related things such as "chunking" and "reasoning by analogy" have been relatively unsuccessful because they did *not* take a proper decision-theoretic stance—of plausible inference and cost-effective action—towards the selection and use of the abstract generalizations that they were making.

In chapter 4, I begin illustrating the use of simple rational abstraction in the game of spider solitaire, which offers some good opportunities for rational abstraction and search-control to show themselves off. Spider also serves as an example of a very large single-agent decision-analysis problem. I will also discuss how to generalize rational abstraction methods to be of some use in other large decision problems as well.

2.9.3 More automated training of evaluation functions

In asking Smith for more details on how he was his producing his evaluation function for his chess program, I was surprised to discover that he was doing a large amount of work by hand: defining a carefully-constructed classification-tree for separating his chess data into bins of approximately equal size, and defining an extremely large set of chess features on which to perform linear fits.

This raises an issue of methodology: as computer game-playing/problem-solving researchers, should we focus a lot of our effort on encoding lots of domain-specific knowledge into programs that introduce new, supposedly general techniques?

I would argue that the answer is no. Groveling over an evaluation function might be the wrong place to focus attention; we should be looking at methods to reduce the amount of human effort required to tailor the evaluation function, and make the computer do more of the work. Besides saving us time, this might give us experience in the use of general machine-learning techniques, which we could reapply when transplanting our rational search methods to other domains.

For example, in the next section I describe a method for constructing good classification spaces automatically from a set of features, by explicitly measuring how well a classification space performs at predicting the data of interest (here, a position's value). We would do well, I think, to use such automated methods when possible, so as to minimize the amount of time we spend writing hand-tailored domain-specific code.

Also, as Matt Ginsberg has argued [Ginsberg-Geddis-91], there are generally domain-independent reasons why domain-specific tricks work. If we expend a little extra effort to find those more-general forms for our computational tricks and techniques, we stand the chance of gaining great benefits when we learn how we, and our agents the computers, can apply that knowledge in other areas.

2.9.4 A note on domains

I would suggest that researchers currently working in typical game-playing and puzzle-solving domains ought to focus more on applying their work to a wider variety of such domains, and even to other, apparently very different “real-world” domains. It’s very easy to get lost in the intricacies of tricks for playing one specific game or another, so much so that we often forget to think about how our tricks might be generalizable. Often they are.

Many people like to work on well-established AI problems, so that the relevance of their work can be easily demonstrated by comparing their new program’s performance with that of well-established, earlier programs. However, I feel that comparison with existing programs in traditional domains can be extremely misleading. These programs usually win not because they have better general intelligence, but because they’re more narrowly focused and specialized to that one small domain.

This is especially true for chess. I think that, in particular, the computer chess community has gone much too far in the direction of domain-specific hacking, for chess competitiveness to be a good measure of the worthiness of, say, a new general theory of search. It’s possible that someday, smart AI programs will beat the brute-force chess engines, but the chess hackers have too much of a head start for chess to be a good place, in my opinion, for an AI researcher to start.

So, if we abandon comparison against programs in well-established narrow domains as a good basis for evaluating AI techniques, how can we evaluate our methods? One solution, proposed by Barney Pell in [Pell-93], is to try to establish a new, broader competitive problem domain; one for which even a program that was very domain-specific would still need lots of general reasoning mechanisms.

Another idea, which is easier because it can be implemented on the individual level, is as follows:

- Strive for generality. Think about several specific domains when designing your method, ones that are as different as you can possibly handle. Make sure that everything you do makes sense when instantiated in each of the domains. The proof of your method will come in showing that it does well in a variety of domains.

- Don't worry about direct competition against other people's programs. This means, don't hack in C to gain competitiveness. Don't spend a lot of time adding domain-specific knowledge to improve competitiveness. If you're working in a well-established domain like chess, ignore the good brute-force programs. You're not going to beat them unless you spend most of your time on domain-specific stuff.
- Efficiency is still important! The realities of limited computational resources should not be ignored. Pay attention to asymptotic running times. Write a variety of simple, brute-force programs for the domain to compare your smart technique against. Show that your smart program does better than the brute-force method even (especially!) when time is taken into account. Be careful though: don't make the brute-force programs asymptotically slower than they need to be, and don't optimize your smart program for efficiency more than you do for the brute-force one. Write your smart program in the same language as the brute-force one. Use the same evaluation function (when possible) for the programs being compared. Don't spend much time on the evaluation function or other domain-specific performance-improvement—unless you think that having more domain knowledge would help the brute-force program more than the smart one, in which case you're on the wrong track. More knowledge ought not to hurt.

2.10 T owards new advances

The traditional state-graph-search approach to game-playing and problem-solving appears to have reached a new pinnacle of achievement in Baum and Smith's technique for rationally and *efficiently* creating and evaluating a partial search tree. Their method remains untested, but theoretically speaking, it seems to nicely wrap of many of the major issues. A number of further refinements of the technique are undoubtedly possible, but it is beginning to look to this researcher as if the larger part of future progress in computer game-playing and problem-solving will be in broadening the scope of the old state-graph-search paradigm. Specifically, I propose broadening it to reason about states at different levels of abstraction. Decision-theoretic tools such as those used by Baum and Smith seem ideally suited for handling the many uncertainties and control issues involved in reasoning at a more abstract level, but many questions remain unanswered. Only time, and research, will tell.

2.11 Automatic Evaluation Functions

This section describes an alternative way to get a good evaluation function from game-playing data, which is proposed as an alternative to Baum and Smith's method of a handmade classification tree and a linear fitting function. This method tries to be as general as possible, and applies to machine learning tasks of all kinds, not just evaluation-function learning.

2.11.1 The Problem

Suppose we are making a game-playing program. We have a game search algorithm that can do the job, if it's given a fast source of a certain kind of probability information. For example, given a game position, we may want to quickly find our personal probability $\wp(Win)$ that we would win if we were in that game position. Or, if it's a game with several possible outcomes $o_1, \ldots, o_n$, we may want a probability distribution $\wp(o_i)$ over these outcomes. Or, for control of search, we may want, for each o_i, some approximate representation of a continuous distribution over the amount that our $\wp(o_i)$ for the position will change after some more computation. In any of these cases, the problem is the same: how do we do a quick, yet high-quality assessment of these probabilities?

One good answer is to get these probabilities by collecting statistics. Unfortunately, in real games there are far too many positions to feasibly collect statistics for each of them individually. The standard solution to this problem is to collect statistics for whole classes of positions at once, without remembering the statistics for the individual positions within a class. The *classification space* for positions is often defined as the cross-product of a number of *feature spaces* $\phi_1, \ldots, \phi_k$, each of which is the range of a *feature* f_i, which is some easily-computed function of game positions (see figure 9, p. 98). We generally try to choose a classification space that we think will provide a lot of information about the variable we are interested in (the game outcome, or our future probability assessment).

Once we've defined our classification space, or set of bins, we can begin gathering statistics from our data. Each data point is assumed to be in the form of a game position P, together with the value of the variable y we are interested in. We classify P, look up a data storage bin for its class, and add y to the data stored in that bin. We might also associate y with some additional information about P that we know relates to y in a certain special way; for example, if y is numeric, we might expect its value to be correlated with the values of certain numeric features $x_1, \ldots, x_k$; these features would then be left out of the feature set used to define the classification space, and instead their values would be stored along with the value of y. (As Baum and Smith do in their position-classifier.)

Then, for each bin, we form a model of the data in that bin, in the form of a probability distribution over y (or, in the case where additional information is stored in the bin, a function from $x_1, \ldots, x_k$ to distributions over y). We might get the distribution by choosing the maximum-likelihood distribution given the data, out of some simple space of approximate representations of distributions.

So then we are done; when we see a game position, we simply look up its bin and use the stored probability distribution as our quick first assessment of our belief about which outcome will actually occur.

Unfortunately, we often find that it's hard to come up with one simple feature that does at well at predicting the outcome. Even combinations of 2 or 3 simple features might not be enough. We generally find that we need a lot of features in order for the statistics to discriminate the different outcomes well. However, this leads to another problem: the more features we have, the larger the classification space becomes (exponentially), and so a huge amount of data is required in order to get a statistically significant sample in each class. This section will briefly describe one possible solution to this problem.

2.11.2 The Proposed Solution

The basic idea is this: even though it may be true that no single feature, or combination of just a few features, discriminates outcomes very well, it may be the case that lots of these small classification spaces do *okay*. If only we could combine the recommenda-

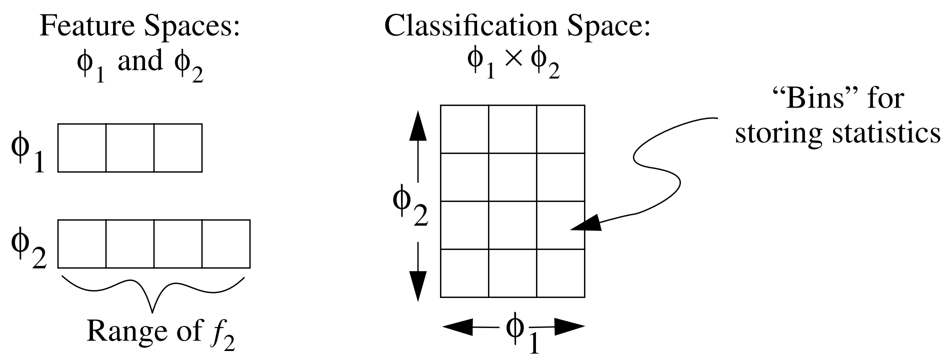


Figure 9. Classification Spaces.

tions of these small classification spaces in a principled way, we might get a good overall answer without having to resort to collecting statistics on the exponentially-large classification space that combines all the features.

For example, we could form a classification space for each combination of 1, 2, or 3 different features. Each data point can be classified within each space, so we only need to collect enough data to get a good sample in each bin of the largest such space.

But when it comes time to use the results, how do we combine the probability distributions produced by a position's classification in all the different spaces? We don't want to just add them together, because some of them will be much better predictors of the outcome than others; we want them to contribute to the answer proportionately. Is there any principled way to weight the distributions?

One possibility is to think of each classification space, with probability distributions in all of its bins, as a probabilistic model of the data, i.e., as a hypothesis that a position's outcome is determined by selecting a value from its class's distribution. Thought of in this way, we can say that each classification space, each hypothesis, has some probability of being the "correct" one. If we could only compute this probability, it would make sense to use *that* as the space's weight. (Assuming that all these hypotheses are mutually exclusive—the damage of assuming this in practice is hoped to be small.)

We can compute model probabilities using Bayes' rule. Since these are probabilistic models, it's trivial to compute the likelihood of the data given the model. So the problem is reduced to one of finding sensible priors for all the models. This is not too critical since, with a good statistical sample in all of a model's bins, the differences in the data-likelihood between the different spaces ought to be the deciding factor.

In any case, the prior for a classification-space model is just the product of the priors for the distributions in each of its bins, since the classification-space can be thought of as a conjunctive hypothesis of the form "in this bin, the distribution is this; and in this other bin, it's this; etc..."

How we compute a prior for the distributions in the bins depends on how we represent the distributions, and whether we have additional stored features (Baum and Smith's $x_1, \ldots, x_k$) complicating our model. However, one answer in the simple case is that the prior ought to be just e^H, where H is the entropy of the probability distribution in the bin; a high-entropy distribution carries less information and so should be *a priori* more probable.

2.11.2.1 Application of the Technique

A version of the above-described technique has been tried in God, a simple card-playing game; the results were very successful.

In God, one player is the "god," and makes up a rule determining what stacks of cards are "allowed." For example, the rule might say that a stack is allowed if and only if the top card has a different suit and than either of the next two cards. The other player starts with all the cards, and he attempts to play them one at a time on a growing stack; after each attempt the god says "yes," or "no;" if "no," the player must take back the card and try again. The player's goal is to get rid of the cards as quickly as possible. Assuming a greedy strategy, his goal on each play is to play the card that forms a stack having the highest probability of being allowed, based on the answers received so far. (Other rules prevent the god from making the rule too restrictive or complicated; I won't go into those here.)

One thing to note is that the data is very scant; in a well-played game the player receives only a few bits per card. Thus my collaborator Mark Torrance and I required a technique that used the data very efficiently. We defined about ten features of card stacks, for example, whether the top two cards were the same suit. We then formed classification spaces for every combination of one or two of these features, and derived predictions from past data as described above. The technique proved very effective in generating good predictions, our player was the best out of about a half-dozen God-playing programs written for a class.

2.11.2.2 Problems With this Approach

I already mentioned the problem that several classification spaces (with bins filled in) might turn out to embody the same probabilistic model of the data, and thus it will be improper to count them as mutually-exclusive hypotheses. This can happen, for instance, with the spaces ϕ_i and $\phi_i \times \phi_j$, if the outcome is totally independent of $f_j(P)$ given $f_i(P)$, because $\phi_i \times \phi_j$ will then represent the same hypothesis as ϕ_i. Thus ϕ_i's contribution to the total distribution will be unfairly magnified. This particular case should not be a serious problem, since $\phi_i \times \phi_j$ will have a negligible probability compared to ϕ_i, as a result of it being a much larger space and so having a much smaller prior. But if two identical features are included by accident, their contribution will be counted twice; and any two features that are not independent will degrade the prediction quality, to a degree that is still unclear.

Another problem is that we are considering only models that contain maximum-likelihood distributions for the data. What if we also considered, in each space, a continuum of nearby models that contained distributions that were slightly different? Might that yield different weights for our classification spaces?

To avoid this problem, we need to make sure that our distributions are not just "most likely" distributions but rather express the sum of all possible distributions, weighted by their probability given the data. If we do this, then we can be sure that when we calculate the probability of a classification space, we are truly calculating the probability of just the hypothesis that it's the right space, not the hypothesis that it's the right space *and* that the distribution in each node is such-and-such particular distribution.[1]

1. But if the hypothesis is just that the space is correct, shouldn't the space's prior depend only on the space itself, and not on the distributions stored in its bins? Maybe its probability should just be proportional to $e^{-\#\text{bins}}$. This needs more work.

If the variable y has a small, discrete range (e.g., $\{\text{Win}, \text{Lose}, \text{Draw}\}$), and we accept a normal flat prior over the space of possible distributions over y, then there is no problem; a standard formula[1] for the exact distribution is given by:

$$\wp(y_i) = \frac{m_i + 1}{M + n_y} \tag{22}$$

where m_i is the number of data points where $y = y_i$, M is the total number of data points $\sum_{j=1}^{n_y} m_j$, and n_y is the number of possible values of the variable y. This formula works well even on small amounts of data. Note that when there is no data, we get a flat distribution; this is the correct weighted sum of all possible distributions if all are equally likely. If there are two outcomes $\{\text{Win}, \text{Lose}\}$ and the data has 1 win and no losses in the bin, when we get $\wp(\text{Win}) = \frac{2}{3}$.

However, if the range of y is continuous (or just too large to store all the m_i's), as is the case if y represents a future probability assessment, then we cannot use (22) directly. In any case, we probably don't want our prior over distributions over such a large space to be flat; smooth, simple distributions seem more likely *a priori* than complex, jagged ones with high spatial frequencies.

One simple method that ignores high spatial frequencies but still allows multimodal distributions is to discretize y into evenly-spaced sections, and use (22) on those. However, this loses the distinction between a needle-sharp peak and a slightly smoother one that's still all in one discrete section.

2.11.3 A More Refined Version

I recently thought of a refinement of this technique that seems like it would produce even more accurate results; however, this new method has not been examined in detail or tested.

In the previous proposal, we weighted a classification space's prediction of a game-position's outcome by the probability that the entire classification space was the "right" model of the data. However, each game-position lies in only one bin in each classi-

1. (22) is the mean of the beta-multinomial distribution, for a uniform prior. The beta-multinomial distribution generalizes the binomial distribution to more than 2 outcomes.

fication space. So perhaps, the probability of that individual bin, rather than that of the entire space, would be the more appropriate quantity for weighting the bin's prediction.

However, this raises a couple of new problems. It complicates the question of whether two alternative hypotheses are independent. Also, it complicates the comparison of the probability of the two hypotheses, since in general they will have been based on different data.

Further work remains to be done on this alternative.

2.12 Towards CGP as a True Testbed for AI

Computer game playing has been called "the drosophila of AI" [McCarthy-90], but have the CGP developments to date really fulfilled that promise? I would say that until recently, they have not. Too much time and effort has been spent on optimizations for particular games (especially chess), optimizations that do not really teach us anything about how to write programs to solve more practical problems. In fact, the entire Shannon paradigm is really a chess optimization. The fact that humans play a number of other games amenable to minimax search is more of an historical accident than anything else. If we broaden our sights a bit to look at other kinds of games, such as games with huge numbers of possible actions, we see that partial state-space search no longer seems so appropriate—especially if the search is unguided.

We can see two things that are fundamentally needed for further progress in automated reasoning: First, we need to have a solid, principled, general understanding of the problem of limited computational resources, instead of just blindly trying arbitrary techniques for control of reasoning and settling on whatever seems to work in one context. Second, we need to question our ontologies, such as the insistence on defining some fixed level of world-states, when it seems that humans shift around, reasoning about game-positions and other problematic situations at many different levels of abstraction.

In the last few sections we have seen that CGP research is beginning to turn to the general tools of decision theory, which at least show promise for understanding the issues of limited rationality. In the next chapter we will look more carefully at these issues.

Nobody, however, has come up with good, general algorithms for game-playing at different levels of abstraction. The problem is that the kind of abstractions needed vary from game to game. In chapter 4 we will begin to explore a general framework for reasoning about abstractions.

However, what is really needed is to combine these two advances: limited rationality and abstraction. If CGP research truly achieves a synthesis of principled management of their limitations with a flexible way of viewing the world, I believe that it both reach new heights of game-playing excellence, and teach us many useful things about reasoning in general.

CGP can still be the drosophila of AI, but it must begin to try consciously to be relevant and general, rather than focusing on next week's chess tournament. Efforts like Barney Pell's Metagame project [Pell-93], which attempt to force CGP work into a more general context, and away from its past focus on the idiosyncracies of particular games, are the right kind of step. Russell and Wefald's attempts to extract general principles of limited rationality from their search algorithms are also good, as are Baum and Smith's application of standard statistical tools. However, whether the masses of CGP researchers will turn to follow some of these new directions remains to be seen. I hope that they will.

Decision theory got its start from von Neumann's investigations of games. In the next chapter we will see how it has since struggled to reinvent itself in limited-rationality terms.

Chapter 3

Limited Rationality

In the preceding chapters we have seen a number of game playing programs. Having adopted the partial state-space search paradigm, the programs answer the question of what part of the space to search in various ways, from a "flat" depth-d search to singular extensions, conspiracy search, and finally the Russell-Wefald and Baum-Smith decision theoretic search techniques. All these techniques are attempting to reconcile the need to make a good decision with the need to search quickly, and this reconciliation requires making the meta-level decision of what to search. But we saw that it was important to be careful not to spend too much time making this meta-level decision, either. So we have a rather complicated set of variables to trade off against each other: quality of move choice, versus time spent searching, versus time spent deciding where to search. It would be desirable to have some kind of theoretical understanding of how to make this trade-off, while keeping in mind that we, as designers, do not want to spend arbitrary amounts of time optimizing our program designs, either. The need for efficiency is really present at all levels of the process of designing, building, and running a game-playing program. We would like to have some kind of higher-level understanding of this efficiency constraint and how exactly it should affect our methodologies, at as many levels as possible.

But before we set forth investigating what would be involved in such an endeavor, it is worthwhile to step back from our domain of games, and consider whether this issue is also important outside that realm—for we should always keep in mind that computer game playing is not our ultimate goal, but is merely a way to focus our attempts to develop better general AI techniques.

Indeed, if we look at the larger context of computer problem solving, planning, and so forth, we find that a similar problem appears throughout, and constitutes a fundamental question in building AI programs, namely, the question of Limited Rationality (LR): How do we efficiently utilize limited reasoning resources, when figuring out the answer to this question also consumes those same resources? This seems at first to be a circular question. And yet, it has been shown (for a particular formalization of this question, in [Lipman-91]) that answers do exist, at least in principle. But the question remains of how to get at them in practice.

This chapter will consider these issues in detail. We will look at some of the related investigations in probability, decision theory and economics. We will also briefly survey the work on limited rationality theory that grew out of Russell and Wefald's experiences with their decision-theoretic search control techniques in game playing. Their work provides many interesting insights, but is not at all the final word on this issue. We will also look at several efforts in other areas of AI to partially formalize the trade-offs of limited rationality. These overviews of existing work will provide the background for our own investigations into the subject.

In my attempt to better understand some of this background, I investigated the issue of whether some of the CGP search control techniques could be applied in another domain, that of clinical decision-making. Instead of large game trees, CDM involves large decision-analysis trees. We will discuss these preliminary investigations in §3.3.

Unfortunately, the outcome of these investigations was the realization that there is a fundamental flaw with the limited rationality techniques developed for CGP, namely that they are (perhaps unsurprisingly) anchored to the same concrete state space paradigm that pervades most of CGP research. When I attempted to generalize the CGP search-control techniques to include a decision-analysis problem, I discovered that the state-space paradigm was not appropriate for that problem, or for reasoning in general. This led to my investigations into a new paradigm of *abstract state-space search*, reported in chapter 4, which attempts to preserve some of the desirable features of the regular state space search while avoiding some of its limitations.

Finally, in the last part of this chapter I present the beginnings of some new theoretical concepts for understanding the problem of limited rationality that are free from the limitations of the state-space paradigm, unlike the formulations used in Russell's group. Whether these concepts can be extended to a complete theory of limited rationality remains to be seen, but at least, they provide an interesting place at which to begin.

Now, let us start by looking at some of the history behind the issue of limited rationality.

3.1 History of LR in Decision Theory and Economics

Economists have long been concerned with the problem of modeling the behavior of agents, groups and societies. Their investigations have often been conducted by modeling the reasoning process performed by these entities in order to choose their actions. In order to keep things simple, these models of rationality often made strong assumptions about the deductive powers of the entities. Such models were found to be unrealistic in ways that damaged the quality of the economic predictions derived from the models.

Thus, Simon introduced the notion of "bounded rationality," the study of models of rationality in which entities may be limited by their internal computational capabilities as well as by the structure of their environment:

> ...some of the constraints that must be taken as givens in an optimization problem may be physiological and psychological limitations of the organism (biologically defined) itself. For example, the maximum speed at which an organism can move establishes a boundary on the set of its available behavior alternatives. Similarly, limits on the computational capacity may be important constraints entering into the definition of rational choice under particular circumstances. We shall explore possible ways of formulating the process of rational choice in situations where we wish to take explicit account of the "internal" as well as the "external" constraints that define the problem of rationality for the organism. [Simon-55]

Led by Simon's ongoing efforts (see [Simon-82]), many economists explored various sorts of bounded rationality models, and these sorts of investigations remain an active area of study in economics and operations research. See, for example, [Arthur-91], [Tietz-92], [Honkapohja-93], [Wall-93], [Norman-Shimer-94], and [Lilly-94]. Unfortunately, we do not have time to survey this interesting and relevant literature here.

Economics and operations research have had close ties with decision theorists since von Neumann and Morganstern [vonNeumann-Morganstern-44] introduced decision theory. While economists were exploring bounded rationality, decision theorists were becoming aware of the difficulties caused by their theories' not taking into account the cost of thinking.

Savage [Savage-67] was one such decision theorist, but he feared that repairing the theory might lead to paradox. Let us look at how this might happen. Standard decision theory says that a rational agent chooses the optimal action, given its preferences and beliefs (this may not be the "truly" optimal action, since the agent's beliefs may be inaccurate). In a limited rationality version of the theory, we might say that the agent is not powerful enough to process its beliefs and preferences completely, so instead we have some model of the limitations on the agent's computational capabilities, and we say that the agent chooses optimally, given those additional limitations. But this seems to be begging the question: if we already know the agent is incapable of making optimal decisions, doesn't this conflict with the assumption that it makes optimal use of its limited computational capabilities? Is this not somehow contradictory?

In [Hacking-67], Hacking responds to Savage's concerns. Hacking proposes that the problem with decision theory is due to the property of logical closure[1] among beliefs in the theory of personal probability used by Savage. Hacking points out that the probability axioms can be expressed in terms of "possibility," and that the theory itself, and all the traditional arguments in support of it, are really independent of the exact definition of possibility that is used. Possibility, in turn, is reduced to knowledge (the idea being that p is possible if it is not known to be false). And finally, Hacking proposes an "examiner's sense of knowledge," in which the objects of knowledge are *sentences* in some "personal language," rather than propositions. Using propositions as the objects of knowledge leads to logical closure because logically equivalent propositions are standardly defined to be identical. In contrast, with Hacking's examiner's view of knowledge,

> ...a man can know how to use modus ponens, can know that the rule is valid, can know p, and can know $p \supset q$, and yet not know q, simply because he has not thought of putting them together. ([Hacking-67], p. 319)

This definition of knowledge leads to what Hacking calls "personal possibility," and to his preferred, sentential form of personal probability. This solves Savage's problem in one

1. Logical closure on beliefs is referred to as "logical omniscience" in the logic community (such as [Hintikka-75]), according to Konolige [Konolige-86].

sense, by making the theory weaker. Now that agents do not make all logical deductions, to restore some of the power of decision theory we would need to model what deductions they *do* make.

But Hacking realized this still left a question unanswered: namely, how are we supposing the agent decides what deductions to make? (This corresponds directly to the question of how a game playing program decides what partial game-tree to examine, which we explored earlier.) Hacking points out that if we don't want to assume an agent acts optimally at the base-level of choosing an action, then we also shouldn't assume it acts optimally at the meta-level of deciding what to deduce—making an optimal decision at that level might be just as expensive, or more, than making the deductions needed for an optimal base-level decision. So it appears that an agent will have to also do some work to "police" its own meta-level deductions—but then, does it not require yet more reasoning to do this "police work," and more reasoning to police the police work? And so on? To get out of Savage's "paradox," would we not have to incur an "infinite regress?[1]"

Hacking does not worry about the infinite regress problem, saying that he is satisfied to just say that the metareasoning stops at some arbitrary level and just makes a fast approximation to the optimal answer. This may be a pragmatic view, but it is a somewhat dissatisfying answer from a theoretical perspective, because it is not clear what a claim of limited rationality would consist of if it did not require that *something* be optimized—either the decision of what to do, or the meta-decision of what computations to do to decide what to do, or the meta-meta-decision, *et cetera*. But at the same time, it appears that any such optimality requirement would require unlimited reasoning powers on the agent's part to assure that it is satisfied. With limited resources, is there any kind of optimality assumption that makes sense at all?

This question was recently answered in the affirmative by Lipman [Lipman-91]. Lipman showed that it is indeed consistent to simultaneously say of an agent both that it is uncertain about the outcomes of its possible options at every level, and that it chooses the optimal action, within the bounds of that uncertainty. This amounts to saying that an infi-

1. Economists have long been wary of this so-called "infinite regress" problem with regard to bounded rationality—Lipman suggests [Winter-75] as an introduction to that literature.

nite tower of levels of suboptimal meta-decisions can be treated as an optimal decision among alternatives at a single level.

However, the model Lipman uses for his proof is very simple (there is no mention of probabilities, for example), and is not intended to be a complete theory of rationality in the tradition of decision theory. It also does not tell us how to find an optimal decision, it is merely an existence proof that there is some optimal thing that an agent can do despite its being limited on all levels. Nevertheless, it is a landmark work and contains many interesting ideas that will undoubtedly be important for future limited rationality research.

Next we will look at some of the limited rationality frameworks that have been developed in the context of computer game playing.

3.2 Efforts W ithin Computer Game-Playing

As I discussed in chapter 2, the problem of limited rationality occurs in a very concrete form in efforts to write programs to play games like chess. In principle, players could just make optimal moves by examining the entire game tree and applying the minimax rule (see §2.4.1). But in practice, this is infeasible, and no easier way to choose optimal moves is known.[1] Thus, programs have traditionally adopted a strategy of approximating optimal play by doing only a partial search. This now raises the issue of control of search, which was traditionally handled in games by doing a flat search. Unsuccessful attempts at search control using "forward pruning" techniques (see [Truscott-81]) were abandoned early on. Later, some *ad hoc* "singular extension" mechanisms specialized for Chess were explored ([Anantharaman-et-al-90], [Hsu-et-al-90]). But more interestingly, as we saw, researchers began to explore general techniques for exploring those parts of the tree thought to be most important for making a good decision ([Rivest-88], [McAllester-88], [Schaeffer-91]). However, these authors were not concerned with the general problem of limited rationality, and did not express what their algorithms were doing in decision-theoretic terms.

In chapter 2 we briefly mentioned Russell and Wefald's CGP work. We now return to examine their work in more detail, because of its intent, unlike most other CGP work, to

1. Except in the few games that have been solved analytically, such as Nim (see [Berlekamp-et-al-82]).

address general issues of limited rationality, in addition to CGP applications in particular. Russell and Wefald's work was the original inspiration for the investigations of rational decision analysis on which I report in the next section.

In a series of papers and a book ([Russell-Wefald-89], [Russell-90], [Russell-Wefald-91], and [Russell-Wefald-91a]), Stuart Russell and Eric Wefald set forth their view of the problem of limited rationality as seen from a CGP perspective, and presented their solution to the problem, in the form of a series of game tree search algorithms.

More important than their particular algorithms, however, is their general characterization of limited rationality, which we will now briefly describe. In their framework, actions that are available to an agent are separated into *external* actions and *computational* actions. Both kinds of actions may affect the current world-state. The world-state is imagined to be divided into external and internal portions; computational actions will generally directly affect only the internal portions, whereas the agent's utility directly depends only on the external portions. The value of a computational action is defined by reference to its effect on the agent's choice of external action.

Russell and Wefald's general characterization of the metareasoning problem focuses on the possible *computations*, i.e., sequences of computational actions, that might precede the next external action. The value of the computation is the difference in value of the actions that would have been chosen before and after the computation.

However, in practice they break down the problem into an incremental form by only considering, at any given time, which computational action to perform next, and making the simplifying assumption (the *single-step* assumption) that the next external action will take place immediately afterwards. (Thus the computations being considered always have a length of 1 step.)

Those are the basic concepts of Russell and Wefald's characterization of limited rationality.

3.3 Application to Decision-Analysis

Expert decision-making systems often evaluate decision trees[1] to determine what action has the best expected utility. But sometimes the decision tree implied by the system's model of the domain is intractably large; not only too large to store explicitly, but also too large to evaluate in a reasonable time. System designers typically avoid such difficulties by simplifying the model of the domain so that it will not product large decision trees, or by abandoning regular decision trees when they become too large and using some other representation to support the decision analysis, such as Markov trees [Moskowitz-84], [Wong-et-al-90], [Sonnenberg-Beck-93].

An alternative, explored here, is to partially evaluate the decision tree, using estimates for the utilities of non-leaf nodes. This is analogous to the standard computer game-playing technique of partially evaluating the typical intractably-large game trees. (In fact, the notion of a *single-player game with chance*, while not the primary focus of CGP research, subsumes decision-tree-based decision analysis in general.)

The problem now (as in game-tree search) becomes two: estimating utilities of non-leaf nodes, and exploring those parts of the tree that are expected to contribute the most to the quality of the overall decision. (See §2.3 for a general discussion of this way of breaking down the problem.)

In CGP, the problem of tree search can be partially handled using pruning algorithms such as Alpha-Beta and Scout.[1] Recently, as we have seen, researchers have gone further, using methods that use information about the degree of uncertainty of utility estimates to select those search actions that will most improve the quality of the overall decision. These methods have met with some success. Especially appropriate are the decision-theoretic metareasoning techniques of Russell and Wefald and similar methods by Baum and Smith, which search those parts of the tree expected to contribute the most to the utility of the final decision. Can the same sort of decision-theoretic metareasoning be profitably used to control the partial evaluation of decision-analysis trees? In my research I have begun to explore that possibility. This section will report on the progress made so far.

---

1. The term "decision tree" is used in decision analysis to refer to a tree of possible future decisions and their nondeterministic outcomes. This is to be contrasted with the usage of the term "decision tree" in machine learning contexts, where it typically means a classification tree of criteria used to categorize different problem instances (for example, in ID3-like systems). In other areas of AI the term "decision tree" is used in still other ways (cf. [Winston-92]). In this thesis we will restrict our use of the term to the decision-analysis context.

1. We have not discussed Scout in this paper. See [Pearl-84] for a detailed description and proof of optimality, and [Bodkin-92a] for a correction of the error in the proof.

3.3.1 Initial investigations

The goal of these investigations is to determine whether decision-theoretic metareasoning, as conceived by Russell and Wefald, could be applied to large decision-analysis problems as well as to two-player perfect-information games, where they applied it. It was noticed that in Baum and Smith's version of decision-theoretic search control, they mention the possibility of allowing chance nodes in their game trees, although the algorithm for this case was not fully developed. However, it seemed likely that the algorithm could be extended to include the chance nodes, and eliminate the second player, reducing the case to the traditional one in decision theory: that of a single person against chance or nature.

I decided that these investigations would require a concrete decision-making domain, within which to conceptualize the details of the algorithm and experiment with an actual implementation. I chose the game of Spider Solitaire for this, because it represents an intractably-large decision-analysis problem, and yet it is easily approachable for research due to its structure being defined by a relatively simple set of rules, as in most games. For more discussion of the motivation for using games, see §2.1 (p. 18).

Once decision-theoretic metareasoning is implemented for this domain, our task will be to compare it with other approaches experimentally, and see whether the metareasoning provides any benefit, in terms of achieving higher average utilities. We may also consider the effect of time, and judge whether higher utilities are still achieved when the time cost is taken into account.

Either a positive or negative result would be interesting. If the result is positive, this opens the door for future work that attempts to apply metareasoning to harder decision problems, perhaps even more "real-world" problems, such as large medical decision-analysis problems (e.g., [Wong-et-al-90]).

If the result is negative, then this tells us something about the limits of the Russell-Wefald limited rationality approach, and may point the way towards generalizations or extensions of the technique that might make it more generally applicable. Or, we may discover some fundamental limitations to this entire style of metareasoning which prevents it from being appropriate for a large class of search problems.

3.3.2 Motivation for Spider Solitaire

I chose Solitaire as my example of a decision-analysis problem for some of the same reasons that many AI researchers have chosen games as a problem domain in general. (See §2.1 for a discussion.) However, the model in decision theory is not typically that of a reasoner and a diametrically-opposed opponent, as is the case in two-player games, but rather is that of a reasoner against nature, a "chance" opponent that is imagined to make moves at random according to some fixed probability distribution. Thus the natural gaming equivalent of decision-theoretic situations is the single-player game with chance. Typical examples are dice games and single-player card games.

Why are single-player card games like dice games? A more natural way to think of card games is as games with "imperfect information," that is, the game has some true state at any time, and the effect of actions is deterministic given this state, but part of the information about the state is hidden from and unknown by one or more of the players. In multi-player card games such as Poker and Bridge, different players have different information, and indeed, in such games, the effect of an action that gains more information (such as drawing new cards) can not be adequately modeled as a chance event with definite probabilities, because due to the fact that different players have different knowledge, they will have different probabilities as to what the cards drawn will be. However, if there is some information about the state that is unknown to all the players (such as the identity of cards yet to be drawn from a deck), the revealing of that information can be taken as a chance event with probabilities determined by the probabilities that would be assessed by a hypothetical observer who knew everything that any player knew. However, since none of these players know what these probabilities would be, it does not pay for them to view the game in this way.

However, in a single-player card game such as a Solitaire game, if there is any hidden information at all, it is of course hidden to all players, and thus when new information is revealed, e.g., cards are turned over, this can be adequately modeled as being chosen randomly from the remaining unseen cards at the time-of-revealing instead of at the time of the last shuffle of the deck. (This makes the uncertainty appear incrementally; whereas reasoning about all the possible permutations the entire deck might occupy would be overwhelming.) Thus, Solitaires with hidden cards can be viewed as nondeterministic games where "nature" or "chance" alternates moves with the human or computer reasoning agent. So a decision tree is an appropriate model of these games.

Many Solitaire card games exist and are described in various books of rules of games (e.g., [Morehead-68]). A number of interesting variations (such as "Seahaven Towers") involve no hidden information at all, and the reasoning situation is just that of a deterministic puzzle, like Rubik's cube or the 15-puzzle, where the initial state is given randomly, but all the operators available have deterministic outcomes, and the problem is to plan a sequence of actions that will, with certainty, achieve the goal. Such puzzles are not so interesting from my viewpoint; I am concerned with reasoning situations that are similar to the kinds of real-world situations where goals typically cannot be achieved with certainty, but more often, only with high probability. I am interested in the problem of figuring out how to maximize your expected utility, and in the metareasoning problem of determining what information to gather to best help one make this decision. Thus, I confine my interest for the present to Solitaires with hidden cards, although ultimately I would like to convert some of my techniques to the perfect-information case as well.

There are also many Solitaires that have hidden cards. Unfortunately, many of them are not very interesting as reasoning problems because there is so little information available to the player, and so little flexibility in one's actions, that it seems that even an optimal strategy would not perform very much better than a relatively naive one. ("Klondike" solitaire is one example.) Thus, in such domains, there would not be much feedback in terms of performance, and large numbers of games would have to be played to determine the relative ordering of two programs' average performance with any statistical significance. Thus, I looked for a game that offered more opportunities for good reasoning to affect the outcome.

The Solitaire game "Spider" is such a game. As Woods says in [Woods-89], "Spider is a particularly challenging double-deck solitaire. Unlike most solitaires, it provides extraordinary opportunities for the skillful player to overcome bad luck in the deal by means of careful analysis and complex manipulations." This is due to the fact that Spider offers many possible ways to arrange existing cards before new ones are revealed, making it more likely that some arrangement will be found that will have a high expected utility upon the revealing of the new information. If not as many moves were possible, it would be more likely that no move would provide a significantly higher expected utility than any of the others. Thus Spider presents the possibility that appropriate use of decision analysis may lead to good performance.

3.3.2.1 Evaluating the method

To apply decision theory to the problem of playing Spider, we require a utility function. I chose to define the utility of a Spider game via a somewhat arbitrary scoring function of the final position achieved.[1] The function is borrowed from a common spider program, and seems to be a good measure of "closeness" to a winning position—i.e., if

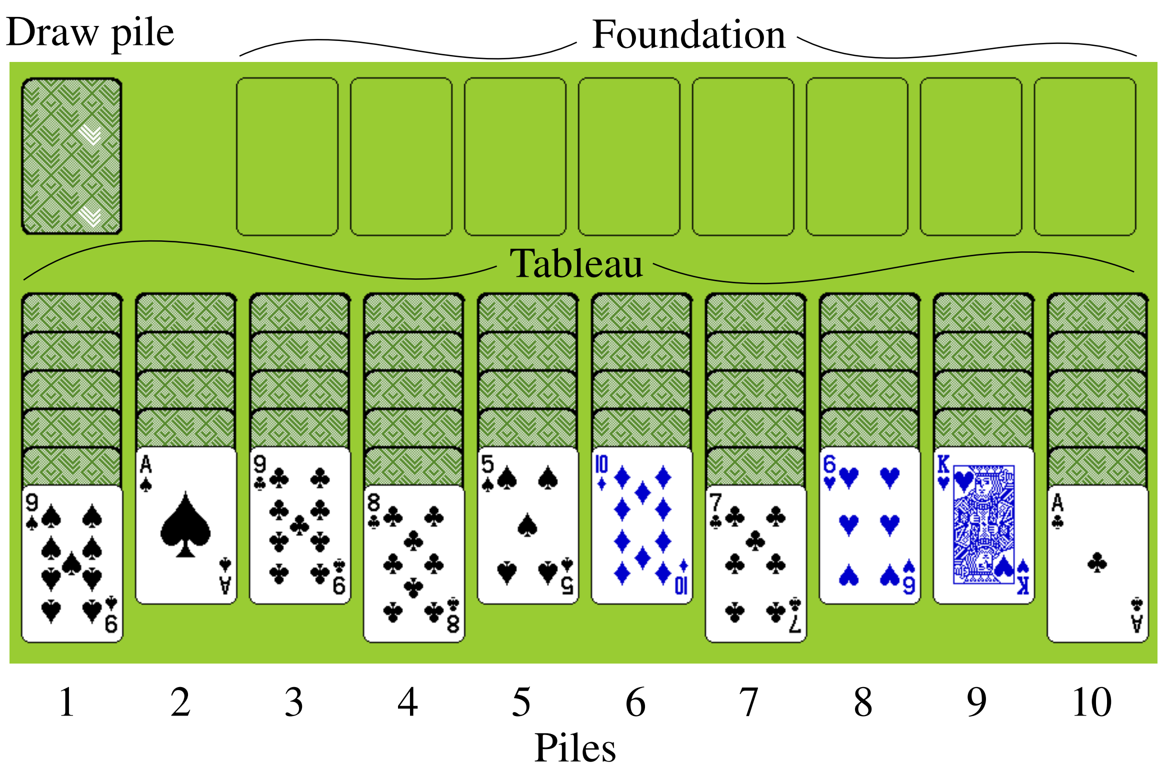


Figure 10. Example of an opening position in spider solitaire.
The moves available in this position are:
Naturals: 8♣→9♣, 7♣→8♣
Others: 9♣→10♦, 8♣→9♠, 6♥→7♣, 5♠→6♥

1. There is an interesting analogy between this measure and the "Quality-Adjusted Life-Years" utility measure sometimes used in medicine (e.g., in [Pliskin-et-al-80], [Perlman-Jonsen-85], [Tsevat-et-al-88], [Hogenhuis-et-al-92], [McNeil-et-al]), which we do not have time to delve into here.

you have a higher average score, then you probably actually win the game more often as well. (Statistics could be collected on this once players are implemented.) The score is defined in any game-state, which allows it to be used as an evaluation function as well. Except in rare cases it is monotonically increasing as the game progresses, so it can serve as a lower bound on the final score that will be achieved. It is thought that this measure will allow "greedy" approaches to be fairly successful. Thus, if a program performs better than a greedy algorithm, it will be doing pretty well.

Using such a scoring function rather than just the win/loss status of the game also allows learning algorithms to have a chance if they start out never winning any games. (In Spider, a random player, for all practical purposes, never wins.)

Our investigations will begin with the construction of various random players. The point of the random player is to establish a minimal baseline such that if we notice that some technique is doing no better than random, we know that the technique is worthless. We will then establish a series of progressively better but still-simple players so that we can compare our final metareasoning player to those as well.

We will generally collect data in the form of a joint distribution over score achieved versus time taken. From this data we can extract many different measures of performance: average score achieved per game, average time per game, average total utility when time incurs an additive linear cost (the amount of which would be an additional parameter), and rate of accumulation of benefits (scores) over time.[1] We can compare the different algorithms under these different measures, giving us more information about the algorithms than if we were to just collect a single evaluative number for each one.

3.3.3 Results to date

In the attempt to characterize reasoning about Spider as a partial search of a decision-analysis tree, and apply Russell-Wefald and Baum-Smith methods to that tree, several problematic issues arose. The first realization was that the nodes in our tree could not represent the true state of all the cards, because that state is unknown. So instead, we

1. This measure is discussed in a machine-learning context by Rivest and Sloan in [Rivest-Sloan-93]. It leads to a significantly different view of the problem, one that is perhaps not so appropriate for medical decision-making. Also one can view Allis's proof-number search [Allis-et-al-91] in this way.

decided to fix the level of our states to be that of “information states” describing all that we know about the state at a given point—all the visible cards. As we mentioned earlier, it is possible to treat the outcome of turning over a new card as if it were a chance event, rather than the gathering of information about the true unknown state. This fits better into the standard decision-analysis model, where we know the outcomes of all the chance events that have occurred so far, but not the outcomes of future ones. (A detailed discussion of how to treat incomplete-information games in terms of “information sets” is currently taking place in the CGP community. See [Smith-Nau-93], [Blair-et-al-92], [Blair-et-al-93], [vanLent-Mutchler-93], and [Gordon-93].)

So using the chance model, Spider is exactly a decision-tree problem. However, when considered in this way, the branching factor is very large: every time one turns over a new card, it may generally be any of however many of the 52 different kinds of cards haven’t had both their instances turned over yet. (There are two of each card initially.) In the early parts of the game, this is still close to 52. And if we really want to do a proper decision analysis, we have to consider all the branches, because our utility (and evaluation function) could potentially be different in each one. (The probabilities, however, can only have one of two values, depending on whether 1 or 2 instances of the given card remain.)

This forces us to face the problem that a partial search of the Spider decision tree will not get very far. And it seems to be the wrong thing to do. Often in Spider, we make moves that turn over new cards, but we don’t care right away what the identities of those new cards are—we may have revealed the card simply as a side-effect of moving some other card in some sequence of moves. But the decision-tree paradigm forces us to separately analyze each possible outcome of the card-revealing “experiment,” even if that card’s identity is completely irrelevant to the next few moves we might make.

One can try to fix this problem by mutilating the concept of a “move” in Spider: you can break a card-revealing move into two parts: the part that uncovers a card, making it available to be revealed, and the part that turns over the card to reveal it. Normally in Spider these are considered one move, because there is never any harm in gaining information earlier rather than later. But breaking it into two allows us to postpone the high-branching-factor card-revealing moves, pushing them farther down into our decision tree, below the part of the tree we will examine in our partial search. This allows our partial search to encompass many more “tactical”-style moves consisting of reorganizations of cards that we could do if we had immediately reveal the new card and repeat our search below that node for each possibility of what the card might be revealed to be.

But playing with the definitions of moves in the game seems like the wrong thing. As human players, we know we can always turn over the exposed cards right away, because getting the information about their identity earlier rather than later never hurts. However, at the same time, when we are *thinking* about a sequence of moves, we are quite capable of ignoring or “abstracting out” the identity of the revealed cards, and thinking about the possible transitions between *abstract* states consisting of specifications for the identity of some of the revealed cards, but not of others.

So we could go ahead and say that we will fix the level of abstraction to be one where the identity of some cards may be unspecified, and declare that a possible move from such a state is to “look” at one of the abstracted-over cards, and incur at that point the branching chance event that allows us to consider separately the different possibilities for that new card.

But now that we are already thinking in terms of searching through abstractions, it becomes clear that a fixed level of abstraction is not sufficient, because there are many other kinds of abstractions about spider moves that would be useful as well. For example, sometimes we want to dissect our thinking about the identity of the new card into just two cases (instead of 52)—is it a King (for example), or is it not a King? Those might be the two natural possibilities to consider in a particular situation when planning what to do after the card is revealed. Another example: often there are two sequences of moves that will leave you in the same board position (transpositions). We want to be able to think about what to do after such as sequence without having to repeat the reasoning for every possible way of getting to the position. (This has long been recognized in Chess and is handled with transposition tables—see [Newborn-89]—and it leads to the exploration of a game graph instead of a tree). But even beyond that: sometimes there are two moves available that can easily be proven to be strategically equivalent, in that, for example, two piles of cards can be switched without changing our description of the known part of the state. The obvious way to handle this abstraction is to canonicalize our pile order so that all states that are identical under such a transformation are treated as a single (more abstract) state.

We can go even farther. In Spider, many moves are reversible, so that a set of positions that is spanned by a graph of reversible links can be thought of as a single, more abstract position, with moves leading from it to other groups of positions that are all connected by reversible moves.

We could, indeed, write a Spider program that includes special-case code to handle all these different ways of abstracting over Spider positions, and write more and more code as new opportunities for abstraction become apparent. However, in this enterprise we would seem to be doing inherently the wrong thing. Wouldn't it be better to generalize all these kinds of abstractions and deal with them all uniformly, instead of making some *ad hoc*, spider-specialized code for each one? For, certainly, we will find similar (but not identical) kinds of abstractions that would be appropriate for other games, and we would like our work on Spider to carry over to those.

So, after investigating these issues, it became clear that a straightforward decision-analysis of Spider would not address the really interesting issues that come up when thinking about how to write a program to play the game. In fact, because of the problem of the large branching factor, a straightforward decision-analysis, would probably perform very poorly compared to a special-purpose Spider program that took advantage of these abstractions, no matter how good the search-control was—simply because in the flat state-space view of Spider, the states are not abstract enough.

Although it would be possible to go ahead and implement the program and perform the experiments on the metareasoning technique, I felt that due to the above realizations, I had already achieved a result with regard to the general applicability of the game-tree search control techniques: namely the realization that those techniques were all designed around a state-space search paradigm, which simply was not appropriate when dealing with a game where many kinds of abstractions are apparent and important at many levels, and the sorts of reasoning that are natural and effective in that domain involve constantly shifting around to different levels of abstraction.

In the next chapter we will discuss how reasoning with abstractions might be dealt with in a way that might be amenable to adapting some of the state-space search control techniques to work with abstractions instead.

In summary, the result of our initial investigations on reapplying LR techniques developed for game-playing to a decision-analysis problem is that the techniques are not directly applicable—even, interestingly enough, when the decision-analysis problem studied was, actually, a game. This points out that fact that even within computer game-playing, the focus of the existing CGP work has primarily been within a very restricted class of games, namely the two-player, perfect-information ones, and that it is surprisingly difficult to reapply the existing CGP techniques to other sorts of formally-definable games.

However, the possibility is left open that with a more intensive effort, CGP techniques, especially the ones using general decision-theoretic metareasoning principles such as Russell and Wefald's, might still be adaptable to different kinds of search domains.

3.4 Other LR work in AI

So our initial investigations into the reapplication of CGP limited rationality techniques to decision analysis were somewhat unsuccessful, although they led to an interesting alternative direction for research. However, it is worth mentioning related efforts in AI to apply LR ideas to practical problems.

Horvitz and his colleagues have long been investigating the problem of how to deal with limited resources in medical decision-making and other contexts. Much interesting work which we do not have time to survey here can be found in their papers; see [Horvitz-88a], [Horvitz-et-al-89a], [Horvitz-et-al-89b], [Horvitz-Breese-88], [Horvitz-89], [Horvitz-et-al-89c], [Breese-Horvitz-90], [Heckerman-et-al-89]. Like myself, Horvitz is applying meta-level control to decision analysis. However, Horvitz's focus is on controlling the refinement of the probabilities associated with the arcs or transitions in a decision tree, whereas my interests lie in refining the estimated or expected utilities at the leaves of a partial decision tree, via deepening the tree in places where the accuracy of those estimations is most in need of refinement.

In the planning community, there has been much work on meta-level control and the LR issues that naturally arise there. The work by Dean and his colleagues is especially central in this area, but again we do not have time to survey it in detail. See [Dean-Boddy-88], [Boddy-Dean-89], [Dean-et-al-93a], [Dean-et-al-93b], [Dean-et-al-92], [Dean-91]. Much of Dean's work utilizes the concept of *anytime* algorithms that can be halted at any time to return an answer of less than optimal quality. Shlomo Zilberstein ([Zilberstein-93], [Zilberstein-Russell-93]) has continued in this line of investigation by examining how to optimally combine anytime subprograms to achieve good overall limited rational behavior.

Recently, Russell and his students have been extending Russell and Wefald's earlier work on general LR theory by formalizing a concept of *bounded optimal* agents ([Russell-et-al-93]). Although it is quite interesting, and will undoubtedly be a good foundation for future work on formal LR, it still falls into a highly state-oriented category of formalisms; whereas my interests lie more in definitions of LR that do not assume a state-space ontology. I will now discuss my ideas along those lines.

3.5 Towards a formal theory

Frameworks for limited rationality such as described in [Russell-et-al-93] are a good start, but they tend to be closely tied to the state-space search paradigm, which as I have discussed, is really inadequate—ultimately we need, instead, to search through a space of possibilities expressed at varying levels of abstraction.

Could we possibly develop a theoretical framework for limited rationality that *doesn't* assume a definite level of state-space structure? One in which time, and world-states, and actions, are really infinitely subdivisible, and it is only the epistemology of our agents, their language of thought, that is discrete? Perhaps this will not turn out to be feasible, but at least, we can conduct some preliminary theoretical explorations in an attempt to find out.

3.5.1 Motivation

We need a theoretical framework for describing *limited rational agents*, agents that are designed with the goal of performing as well as possible, given a fixed level of computational power, when time is a factor in performance.[1] Such agents are in contrast to agents that are designed to find optimal solutions to problems, regardless of how much computing time will be consumed. As described earlier, Russell and Wefald have argued persuasively in support of the limited-rationality approach, and they have begun to develop theoretical tools for formalizing what limited rationality means, and for explaining the behavior of limited rational agents.

In this section we begin to build the foundations for a somewhat more general (and, we think, ultimately more useful) theoretical framework for describing limited rationality, and we show how various concepts defined in the Russell-Wefald framework, and other frameworks, can be expressed within the new framework.

3.5.2 Probability notation

There are many formalizations of probability theory in use today. Although the differences between them are largely just unimportant mutations of notational syntax, we feel that in order to achieve complete clarity when talking about probabilities, it helps to set forth the exact, formal system and notation that we will be using, rather than just assume that all readers are familiar with the particular syntax being used. This will also give us opportunities to define some convenient, abbreviated notations for certain probability concepts that we will use especially often.

Our probabilities, events, and random variables will be defined over a space Ω of outcomes (sometimes called a *sample space*). For now, our definitions will require that Ω be finite; the infinite case requires more care in the definitions, but in either case the theoretical tools for limited rationality that we will build from them will be used in the same way.

Definition 1: A *distribution* p is a function $p\colon \Omega \to [0, 1]$ from the outcome space Ω to the real interval from 0 to 1, subject to the standard probability-theory constraint that

1. Of course, some entities have the option of increasing their computational power, in which case they are faced with a broader trade-off, between making better use of their computational resources, or paying the costs of amassing more. For example, an AI research group may buy new computers instead of improving their algorithms, or the industrial economy as a whole may choose to develop faster semiconductors, rather than finance extensive research into limited rationality theory.

$$\sum_{\omega \in \Omega} p(\omega) = 1. \tag{23}$$

We treat a probability distribution as a function rather than as an operator (compare with the more-standard $P\{\omega\}$ notation) because we will want to consider many different distributions $p, q, r, \ldots$ computed by different agents at different times.

Definition 2: An *event* $E \subseteq \Omega$ is a set of outcomes.

Although distributions are function of outcomes, it will also be convenient to define the result of applying a distribution to an event, as follows:

Definition 3: Let p be a distribution, $E \subseteq \Omega$ an event. Define $p(E)$ to be $\sum_{\omega \in E} p(\omega)$.

The following is the standard definition of conditional probability.

Definition 4: Let p be a distribution, and let E and C be events. Then define the *conditional probability of E given C under p*, written $p(E|C)$, to be $p(C \cap E)/p(C)$, i.e., the probability that the event C occurs and also the event E occurs, divided by the total probability that the event C (called the *condition*) occurs (regardless of whether E occurs or not). If $p(C) = 0$ then $p(E|C)$ is undefined.

Unfortunately, there is no standard, concise way to denote, for a given condition Y, the corresponding function mapping events to their conditional probabilities given Y (although this function can be denoted using lambda calculus as $\lambda X.p(X|Y)$). We now introduce a convenient notation for this concept:

Definition 5: Let p be a distribution, C an event. Then we define p_C to designate the distribution *p given C*, defined as

$$p_C(\omega) = \begin{cases} 0 & \text{if } \omega \notin C \\ p(\omega)/p(C) & \text{if } \omega \in C \end{cases}. \tag{24}$$

Note that if $p(C) = 0$, then p_C is a partial function; in fact, it is undefined everywhere. Using this definition together with definition 3, we can designate the conditional probability of event E given event C under distribution p as simply $p_C(E)$, where p_C is the desired function that maps events to their conditional probabilities.

Definition 6: A *random variable* V is a partial function $V: \Omega \to R$ from outcomes into some range R.

We do not restrict R to be a numeric range as in some other formalisms; however, certain operations (for example, the expectation operator, definition 8) over random variables will only be defined in the case where R is numeric.

We also define a new notation for the event that a random variable applied to the outcome has a particular value.

Definition 7: Let V be a random variable, and let $x \in \text{range}(V)$. Then the expression $V = x$ denotes the event $\{\omega \mid V(\omega)\text{defined} \wedge V(\omega)=x\}$.

Definition 8: Let V be a *real random variable*, that is, a random variable with a real-valued range, and let p be a distribution. Then the *expectation of V under p,* written $\text{Ex}_p[V]$, is given by:

$$\sum_{(\omega \in \Omega) : V(\omega)\text{defined}} p(\omega) V(\omega). \tag{25}$$

Additionally, it will also be convenient for us to define the meaning of applying a real random variable to an event, subject to a distribution:

Definition 9: Let V be a real random variable, E an event, p a distribution. Then $V_p(E)$ will denote $\text{Ex}_{p_E}[V]$, the expectation of V under p given E

The point of this definition is that we will later define utility just to be some real random variable U whose expectation we happen to be trying to maximize, and so definition 9 will automatically let us express the utility of an event E (subject to a particular distribution p) by $U_p(E)$.

We also define a special notation for building random variables out of other random variables:

Definition 10: Let $V_1, V_2, \ldots, V_n$ be random variables, and let $e(v_1, v_2, \ldots, v_n)$ be an expression containing $v_1, v_2, \ldots, v_n$ as terms. Then $\langle e([V_1], [V_2], \ldots, [V_n])\rangle$ will denote a random variable defined as follows:

$$\langle e([V_1], [V_2], \ldots, [V_n])\rangle(\omega) = e(V_1(\omega), V_2(\omega), \ldots, V_n(\omega)). \tag{26}$$

For example: Let X be a real random variable. Then $\langle \log [X] \rangle$ denotes a random variable whose value for a given outcome is the logarithm of X's value for that outcome.

3.5.3 Domain Languages, Information States, and Agents

Our goal in this chapter is to provide a formalism for talking about limited rational agents. To talk about an agent's rationality, we need to have some way of describing its what the agent is supposed to be doing, what domain it is working in.

We will assume there is some set W of the possible world-histories of the agent's domain. We will use W as the outcome space for talking about probability distributions. We also assume there is some language L, called the *domain language*, for talking about the domain in which the agent is working (sometimes called its *micro-world*). The language L is a set of sentences that make assertions about the domain. Each sentence $s \in L$ has some content, namely, the set of possible world-histories of the domain that are consistent with the sentence's being true, i.e., the event that the sentence is true. Let $M: L \to 2^W$ be the meaning function mapping sentences to their content.

We note that for agents that do metareasoning, the problem domain concerns not only the part of the world that is external to the agent, but also the internal computations of the agent itself. Thus some sentences in L may make assertions about the agent's internals.

Definition 11: An *information state* of an agent is a set of sentences $S \subseteq L$. Intuitively, an information state represents the information held by the agent at some time.

This is not to say that an agent necessarily represents its information as a list of sentences; just that we can describe an agent's information in this way.

Definition 12: The *content* of an information state S, written $M(S)$, is the intersection of the contents of all the sentences in S:

$$M(S) = \bigcap_{s \in S} M(s). \tag{27}$$

Thus, the content of an information state is an event as well; the event that all the pieces of information (sentences) in the information state are correct.

Definition 13: Let X and Y be information states or sentences. We say X entails Y (written $X \rightarrow Y$) iff $M(X) \supseteq M(Y)$.

We do not assume that an agent necessarily has access to the content of any information states or sentences, and we do not assume that the agent is aware of any particular entailment relations between them. The agent's information state is not assumed to be closed under entailment.

Generally, an agent will gain information over time, which will mean that the size of its information state will increase, and thus the size of the content of its information state will decrease, because fewer world-histories will be consistent with the agent's increased knowledge.

Now, all this is not to say that the agent will actually store its knowledge by explicitly listing the set of world-histories in its information state. Rather, this is just a theoretical tool that will be useful for explaining what a limited rational agent is doing, particularly one that works by deliberative metareasoning, i.e., by explicitly considering different acts of reasoning that might be performed. We are not here talking about what kind of knowledge an agent might have or how that knowledge is represented, we are just saying that whatever an agent's state of knowledge, there is a corresponding event.

Similarly, for each action that an agent may perform, there will be a corresponding event—the event that the action is performed, the set of world-histories in which the action is performed.

Most formalizations of agents, such as Russell and Wefald's in [Russell-Wefald-91a], give the domain more structure than just saying it is a set of possible histories. Namely, they say that the world of the domain outside of the agent is always in some discrete world-state, and that the world passes through a sequence of states in response to an agent's actions. This kind of structure will be important for us later, but for now we can proceed without it, and just talk about events.

3.5.4 Rational Interpretations of Agents

A rational interpretation of an agent is a way of ascribing beliefs and values (probabilities and utilities) to the agent that makes all its decisions appear to be rational. Under a rational interpretation of an agent, the agent is always acting so as to maximize the expectation of its utility under its distribution.

Formally, we will say that, relative to the outcome space Ω and domain language L, a *rational interpretation* R of an agent α is a pair (p, u) of a distribution p over Ω, and a random variable u over Ω, such that whenever α is in information state $S \subseteq L$ and is required to immediately (without changing its information state) choose between mutually-exclusive actions $a_1, \ldots, a_n$, it will choose an a_i that maximizes $u_{p_{M(S)}}(E_i)$, where $E_i \subseteq \Omega$ is the event that action A_i is the one chosen.

Thus, we will say that for any limited rational agent, there is, implicit in its design, some distribution p over Ω, and a random variable u over Ω, and that they have the following property: when the agent is in knowledge state K, and it is forced to immediately commit to one of a set of mutually-exclusive actions $A_1, \ldots, A_n$, it will commit to the A_i that maximizes $u_{p_{M(K)}}(A_i)$, i.e., the one with maximum expectation of u under p given K. When we observe the agent committing to an action A, we explain this by saying that p and u must be such that the utility of A under p given K is higher than that of the other actions to which the agent might commit itself.

As analyzers of a particular rational agent, we may not know exactly what the functions p and u are, and the agent itself will not in general know exactly what they are, in the sense of being able to give, for any ω, the values $p(\omega)$ and $u(\omega)$. But we surmise that the p and u exist, because that is useful for our model. If the agent is deliberative in its rationality, it may be able to report *some* information about p and u, namely, that information about p and u that it is explicitly computing and using.

To emphasize again: we are not presuming that an agent actually chooses actions by computing the sum of the product of probability and utility over all the world-histories in the event of that action being performed. Nothing of the sort. Rather the story is as follows.

Someone gives us an agent. They claim it behaves rationally. What does this mean? This means it behaves rationally relative to some p and u. Which p and u? We may not know exactly, but we can deduce constraints on p and u by observing what decisions the agent actually makes.

For an agent that is claimed to perform deliberative reasoning, we can do more: it will work explicitly with some probabilities and utilities, and when it does, it will enable us to deduce extra information about the functions p and u.

3.5.5 Limited Rational Agents

Relative to a fixed W, L, and M, we say that an agent α is a limited rational agent, if, for any rational interpretation $I = (p, u)$ of α, α fails to take full advantage of its information when constructing its probabilities and utilities. For example, in a limited rational agent, it may be the case that its p assigns nonzero probability to some world-histories that an agent with unlimited computational power could prove were impossible.

The key to handling limited rationality within standard decision theory is to accept that a limited rational agent will assign some nonzero probabilities to some impossible world-histories. That is, we do not assume that our outcome space ω consists only of possible world-histories; it may include impossible ones as well, and p may assign them nonzero probability. For example, suppose the agent is about to perform a certain deterministic computation. There is only one possible outcome. But in advance of the computation our agent will not know what the outcome will be. Thus our agent can be seen as assigning nonzero probability to an impossible outcome, namely that the computation comes out differently than it actually will.

3.5.6 Notation for describing metareasoners

In our framework, we will be describing various agents that explicitly compute various probabilities and utilities for use in their decision-making. For each agent, we will interpret what it is doing as follows:

For each agent α, we imagine that there is some distribution p_α which is the distribution over outcomes that is implicit in the design of α. Now, the agent α, having limited resources, cannot possibly know or compute its probability $p_\alpha(\omega)$ for every possible $\omega \subseteq \Omega$, because the space of possible world-histories Ω will generally be much too large. Instead, α will generally be computing the probability $p_\alpha(E)$ of some event $E \subseteq \Omega$, or even the conditional probability $(p_\alpha)_B(A)$ of an event A given another event B. Or perhaps the agent is computing the expected value $\mathrm{Ex}_{p_\alpha}[E]$ for some event E. But rather than requiring that the agent compute such values constructively from $p_\alpha(\omega)$ values, we *interpret* the values the algorithm computes as having meanings such as the above, and we surmise there is some p_α "built into" the agent that is consistent with all the computed values. The computed values can be seen as empirical data placing constraints on what the agent's "actual" distribution p_α might be. In other words, we are using probabilities as a way of explaining and interpreting what the agent is doing; we are not providing rules for how the agent must compute its probabilities. The method of computation will vary among the different reasoning algorithms.

Now, what of change over time? Generally, an agent's beliefs will change over time. One way to handle this is by indexing the distribution by both the agent and by the time: $p_{\alpha, t}$. However, this does not provide us with any structure relating our agent's distribution at one time with its distribution at another time. A better way is to imagine that for each agent there is but one distribution p_α, but that, after our agent discovers new information, then when we talk about the probability of an event E, we are implicitly referring to the conditional probability of E given the event I that expresses all the information that α has discovered so far, i.e., we mean $(p_\alpha)_I(E)$.

Now, moreover, we will treat computations themselves as information-gathering operations just like those that gather information about the outside world, as do Russell and Wefald. So when we perform some computation that enables us to assess a more refined value for the probability of some event E, this does not mean that $p_\alpha(E)$ has changed, rather it means the following: let C be the event that the particular computation in question came out in the particular way that it did. Before the computation, when we referred to the probability of E, this meant just $(p_\alpha)_I(E)$, where I is the event representing all the information we had gathered so far. After the computation, when we refer to the probability of E, it means $(p_\alpha)_{I \cap C}(E)$, because we have now also gained the information that C holds. It is consistent to have different values for $(p_\alpha)_I(E)$ and for $(p_\alpha)_{I \cap C}(E)$, at least, it is if $(p_\alpha)_I(C) \neq 1$. Generally, we do not expect the agent's distribution p_α to contain perfect information about what the results of all our computations will be. The results of all our computations could in principle be figured out by a reasoner with unlimited resources, but p_α is not intended to model the outcome probabilities that an ideal reasoner would assess, rather it represents the probabilities implicit in the design of our limited rational agent, who is unable to compute in advance what the results of all his future computations might be.

This perspective on probability and computation allows us to formally make sense of various concepts that metareasoning researchers have heretofore used informally. For example, Baum and Smith's agent uses distributions over what its probabilities will be in the future after doing more computation. In particular, they have, for leaf nodes in their game tree, distributions over what the probability of winning will be assessed to be at that node after that node is expanded and "more search" is done below it.

In my framework I can formalize this concept in terms of the following entities:

- p, the distribution for our agent
- I, the event expressing whatever information we currently have.
- N, the event that a given game-tree node (board position) is reached in the actual game.
- W, the event that we win the game.
- E, the event that we indeed expand the node in question before making our next move.
- C, a random variable for all the computational events that will occur before we make our next move. In other words, for a given world-history ω, if $\omega \in I$

(i.e., if ω is consistent with what we know so far), then $C(\omega)$ is defined to be the event expressing all the information that we know, in that history, at the time the next move is made. For $\omega \notin I$, we leave $C(\omega)$ undefined.

Given these entities, we can express our current value for the node by just $p_{I \cap N}(W)$—the probability of winning, given everything we know so far and given that we do eventually move to the node in question.

Now, using definition 10, we can define a random variable V for Baum and Smith's notion of "the future probability of winning that we will assess after gathering more information," as follows:

$$V = \langle p_{[C] \cap N}(W) \rangle. \tag{28}$$

V is the desired function mapping world-histories to the probability of winning we assess for the node at the time of our next move, in that world-history. Note that V is a partial function that is undefined wherever C is, namely, for those world-histories outside of I. This is okay since we only care about world-histories that are consistent with what we know so far.

So when Baum and Smith assess their distributions over future values of nodes, they are computing values of the function $\lambda x . p_{I \cap E}(V \Rightarrow x)$. That is, a distribution over the future value of the node, given everything we know and given that the node is expanded.

What I mean is that when a Baum-Smith agent assesses such a distribution, we can understand and formally describe what it is doing in that way.

3.5.7 Knowledge-relative utilities

Decision analysts have found that when people are required to assess their utilities for different outcomes, they do not give one consistent set of values; their assessments change from one moment to the next [Tversky-Kahneman-87]. We could attempt to explain this phenomenon by saying that people do not report their true utilities accurately, and instead their reports are noisy readings of their true utilities. However, experiments intended to measure utilities directly using lotteries showed the same effects. I think that a fairly obvious introspection shows that we really do change our evaluations of possibilities from one moment to the next, and that this effect is not just a difficulty in reporting the "true" utilities presumed to lie inside us. The reason we change our utilities is also obvious, namely that as time passes we think of more ramifications and consequences of the outcome in question, which in real life is never really a total, final outcome, or a complete world-history, but instead always has future effects that bear contemplation.

However, one may choose to model the situation by saying that for each individual there *are* definite utilities that the individual would arrive at if he were suddenly to become omniscient, or to have infinite reasoning and computation capabilities, or some such, and the utilities an individual reports are simply his estimates, based on the information and thinking he has attained so far, of what *his true* inner utility for the proposed event would be.

But for my purposes, I would like to pursue a different approach. It seems implausible to me that what a person reports as his values are a mere estimate of that person's true inner values that neither that person nor anyone else can ever arrive at. It seems to turn a person's moment-to-moment values into a mere epiphenomenon, when in reality they are all that we can ever really deal with. So instead, I propose to treat a person's moment-to-moment values as the fundamental, primitive entities in my further discussions of utility. How, then, are a person's changes in values to be explained? Well, I propose that a person's *utility structure* be a function from possible states of knowledge to utility functions over outcomes, which give the utilities that the reasoner would assess for the outcome in that state of knowledge. (If our model of the reasoner is not exact, we might also wish to generalize this to be a probability distribution over utility functions, where the actual utility function in a given knowledge-state is chosen at random according to the distribution.) Formally:

Definition 14: Let S be a universe of possible states of knowledge, L a language of outcomes, and α an agent. We write $U(\alpha)$ for *the utility structure of* α, which is a function giving, for each knowledge state $s \in S$, the *utility function of* α *in* s, written $U(\alpha)(s)$, which is a function from outcome descriptors $l \in L$ to utility values $U(\alpha)(s)(l) \in \mathbf{R}$.

So, we imagine that although α cannot change its utility structure, it remains true that since α's state of knowledge s is constantly changing, the current utility function $U(\alpha)(s)$ may in general change as well.

This gives us a framework in which we can describe particular types of agents, such as "rational" agents, that change their utilities in particular ways when they obtain new information. Our descriptions of types of agents can now be formalized as restrictions on the utility structure $U(\alpha)$.

This approach is incompletely developed, but it perhaps points at some useful directions for future work.

3.6 Directions for future development

It is still unclear which of the above ideas concerning LR formalisms will prove most fruitful. I think that we have not quite arrived yet at the appropriate foundation on which to build an LR theory. However, in the above efforts we are getting closer to success. One qualification is that we need to give more thought to deciding exactly what will be the nature of the entities that will be the objects of belief and utility. For example, we have already decided that we do not wish to assign utilities only to states at single moments in time; assigning them to world-histories would be more flexible. However, when we consider the limited nature of our agents, we can see that focusing on concrete, completely-specified histories as the primitive objects of our agent's beliefs is assuming too much; the agent can really only deal on a level of partial, incomplete *descriptions* of sets of outcomes, and can only assign probabilities and utilities to those. Perhaps we want to think of these descriptions as something like Konolige's *epistemic alternatives* [Konolige-86], in that they do not correspond to possible worlds, or "impossible possible worlds," rather they correspond to worlds that are possible *as far as the agent knows*. This hearkens back to the concepts of *personal possibility* discussed by Hacking ([Hacking-67]). Also, they should perhaps be used in a way similar to Lipman's use of states, in which the set of states can always extended to encompass a larger number of more refined states that take into account additional considerations about the world.

So what are these objects we are trying to get at, exactly? Are they syntactic entities? We perhaps do not want to assume or attach a particular language to them; we may want them to have identities apart from the particular language we choose to express them. However, it is difficult to see exactly how to do this. So instead of saying what the epistemic alternatives are themselves, perhaps we should just say that we are just always working with some set L of unique *labels* $l \in L$ for each of our epistemic alternatives (hereafter EA's), whatever they are. This is similar to how some treatments of thermal physics avoid discussion of what states really are by working with sets of labels for states instead. We do not care what these labels are, or what the language is like that we use to label the epistemic alternatives. However, in contrast to the case in physics, we want our epistemic alternatives to correspond to abstractions, in that some may be more or less specific than others; some may subsume, or be implied by, others. We could do this by mapping each EA to the set of world-histories to which it corresponds, and then using the subset relation, but again, we are trying to get away from even talking about a fixed level of concrete, completely-specified things. So to be more flexible, perhaps we can just say that there is some partial ordering on L that expresses the subsumption relationship. Of course our agent will not in general know this ordering, i.e., it is not true that for all pairs of labels (l_1, l_2), he knows whether the EA labeled by l_1 entails the EA labeled by l_2. The various entailment propositions, and their negations, may even be epistemic alternatives themselves.

Armed with our concept of epistemic alternatives, we can rephrase all the definitions presented in the previous subsections in terms of them, as opposed to the existing explication based on sets of world-histories. We can have probabilities and utilities for each of the epistemic alternatives, indexed by the agent and the time of reasoning, and whatever else we want. We can interpret various EA's as being explicitly considered by an agent when it constructs internal representations for them, similarly to how we considered a game state to be "examined" when an agent constructed a representation for it. The explicit EA's might have a special status in our description of what in agent is doing when it is reasoning.

However, a theory based on such a foundation might be too weak. Also, in these efforts we are a long way from doing anything that will effectively constrain our search for good, practical, limited-rationality architectures and algorithms. But let us reemphasize that this work is of a very exploratory nature; we are examining a wide range of possible theoretical foundations in hope of someday finding a better, more general theory of rationality. To do this, we are considering abandoning some of the traditional restrictions on the nature of the objects of belief and utility (i.e., that they are propositions, or sentences, or complete world-states, or sets of such), and in doing this, we are perhaps ignoring some of the motivations that led to those restrictions originally. But a priori, we cannot know that those restrictions will not lead to unrealism or inaccuracies in a theory of reasoning based on them. Indeed, we saw that treating belief as concerning propositions can lead to an unrealistic consequence, namely logical omniscience.

But the more generalized concept (of epistemic alternative) that we are examining here, while perhaps being general enough to build just about any model of reasoning upon, has not so far constrained us to the point of finding what the good models are. To make further progress, we will perhaps need to experiment with more specific and constraining theories, even if they might be considered arbitrary, to see if they will get us somewhere.

At a broader level, one might ask whether perhaps in psychology and the social sciences this sort of exploration will ever work—perhaps good models are inherently harder to find in those fields than in physics, say, because the important concepts in those realms have a very "fuzzy" nature (whereas the important quantities and entities in physics have a more clearly definitive nature).

On the other hand, just because success in rationality theory (and other areas related to theoretical AI) is difficult doesn't mean we should give up. The benefits from explorations like this could be huge. We are, after all, a wealthy society (compared to those that existed in earlier eras) and can afford to spend some time seeing whether techniques of mathematical modeling that worked well in, e.g., physics will also work well for the perhaps less clear-cut matters of agents, societies, economies, beliefs, preferences, rationality, and so forth.

Now, in the next chapter we will go on to look more carefully at some of the abstraction issues that were suggested by our investigations of how to apply limited rationality techniques to Spider solitaire.

Chapter 4

Abstract State-Space Search

In the last chapter we saw that the attempt to extend some of the limited rationality techniques for game playing to a more general decision-making context (including more general types of games, and general decision analysis) fails to immediately succeed due to the general inadequacy of the state space paradigm. To reason about a microworld solely by reference to a graph of its "concrete" states and state-transitions is to shut out the possibility of finding some higher structural level on which to interpret and reason about that world. Thus, we would like to at least explore the possibility of moving away from this focus on concrete state spaces, both in the development of our theories of limited rationality and in the application of those theories to more general sorts of decision-making systems.

Nevertheless, the state-space paradigm has some attractive features. It is easy to visualize and understand. It allows us to apply all the theoretical mechanisms of graph theory in reasoning about our problem. It is conducive to developing simple techniques for collecting and processing knowledge, namely via expansion of partial graphs and propagation of information through the graph. Could we possibly preserve some of these desirable features as we move towards a new paradigm?

In this chapter we begin to explore this question. Like other researchers (e.g., [Sacerdoti-74], [Wellman-88], [Knoblock-91], and others), we arrive at the notion of using a graph of "abstract" states instead of the "concrete" ones used in earlier work on Computer Game Playing (see §2.3) and Planning. However, the current work differs from the earlier ones in that it forgoes including abstract states together with the concrete ones, and considers instead banishing the traditional ontology of states and transitions entirely, in order to achieve more generality. I believe that it is possible to reason effectively about a microworld, or the real world, without necessarily defining some fixed level at which things "bottom out" in atomic states. (This was discussed towards the end of the previous chapter.)

Next, we begin exploring how to generalize some of the many algorithms for effectively growing and evaluating partial state-graphs that we saw in chapter 2, so that we can reapply them to the problem of exploring and processing the abstract state graphs that we will be defining.

The work so far on this topic is preliminary, so at the end of this chapter we step back and evaluate what has been accomplished so far, to see whether this is a worthy direction for further pursuit. We conclude that it is, but that there are other problems that may need to be solved first.

4.1 Planning and game-playing

Planning is reasoning about a hypothetical or real future situation that facilitates making an informed recommendation (a plan) about what to do in such a situation, possibly with justifications, for use by some agent (often the same agent that is doing the planning). Planning is necessary because we want our agents to perform well, and since we generally don't have an oracle that can always tell the agent the right thing to do on demand, at some point reasoning must be performed to figure it out. Once a plan has been made, it can serve as a sort of substitute for the nonexistent oracle, quickly telling the agent what to do.

Games serve as useful microcosms for testing planning techniques, in lieu of more complex "real-world" domains. The games used in AI research are generally characterized by simple, well-defined rules[1]. Such games can still be very rich environments, involving factors like hidden information, chance events, other agents with unknown strategies, and the passage of time. Thus, games offer opportunities for testing solutions to many of the problems that occur in planning in real-world domains.

Unfortunately, the potential of games as a domain for planning remains largely unrealized, because most of the existing work in computer game-playing has made a particular, very strong set of assumptions about what kind of reasoning will be performed by

1. People also play games having complex, incomplete, and ill-defined rules, but such games are much more difficult to use as testbeds for AI research.

the programs doing the planning. These assumptions restrict the range of games that can be handled appropriately by the reasoning technique, and limit its general applicability towards other kinds of planning. They also, I believe, hurt performance compared to what could ultimately be achieved by more general reasoning methods.

4.2 Restrictiveness of current CGP methods

An important characteristic of any approach an agent might take to solving a problem is the ontological stance the agent exhibits towards the problem domain; that is, its idea of the fundamental objects and properties that exist in the domain. Traditionally, computer game-playing researchers have almost always adopted the *state-transition ontology* for understanding a game. In this ontology, the micro-world of the game occupies some definite *state* at any time, out of a well-defined *state-space*; change in the world occurs via well-defined *transitions* between states.

The state-transition ontology is natural, given that we CGP researchers only intend to address games having well-defined rules, and often, these rules themselves define the game in terms of states and state-transitions (such as an arrangement of pieces on a board, and a set of legal moves). Game designers and players find games based on states and transitions easy to describe and think about. Also, in computer science, we have a natural representation corresponding to states and transitions: the graph, with its nodes and arcs.

However, adopting the state-transition ontology entails many consequences (some perhaps unforeseen) for the kinds of reasoning done by our game-playing programs.

One consequence is that, in our attempt to keep things simple, we tend to think about the state of the world as consisting *only* of the game-position as discussed in the rules, when other properties of the world may also be important to the task of planning good moves; for example, the passage of real time, the mental states of the players, and the history of past game-positions. As a result, the reasoning techniques we develop tend to neglect those properties, and they end up only being addressed in an ad-hoc way, or not at all.

Additionally, when we focus on states and state-transitions as the fundamental entities, we tend to develop algorithms that work by taking advantage of this simple structure; by traversing the state-graph, examining individual states. As a result, we tend to restrict our attention to the limited class of games in which such exploration is not obviously inappropriate. For example, here are some cases in which state-graph exploration is drastically inappropriate. In each of these, the need is apparent for a less limited, more abstract way of reasoning about the problem:

- A card game with lots of hidden information, where the current state could be any of an astronomical number of possible states. Instead, it might be better to abstract out the hidden information, and focus on the state of the cards we can see.
- A continuous-motion video-game, in which important changes happen only gradually over many dozens of state-transitions. Instead, we might want to abstract over individual ticks, and instead focus on large-scale transitions that involve many atomic actions.
- A war-game with simultaneous motion of dozens of pieces, so that the number of possible transitions (or branching factor) from any state is astronomically large. In this case, it seems necessary to abstract over many of the details of moves being considered, and instead refine a partially-specified plan of action.

I claim that the same kinds of abstractions whose necessity is obvious in the above example are also needed in traditional CGP games like chess, in order to achieve really efficient use of reasoning resources. The need for abstraction in those games has simply been less obvious, because the state-space is barely simple enough so that the amazing speed of our computers is sufficient to allow brute-force state-space methods to work fairly well. But I predict that once abstraction methods have been studied for as long as the brute-force methods have been, abstraction methods will easily beat brute-force methods, even in games like chess, when given the same computational power.

4.3 Abstraction in game-playing and planning

Abstraction in planning and game-playing is not a new idea. In fact, existing game-playing programs use abstraction in several ways. The common notion of game "state" abstracts out the time, the history of previous states, and the mental state of the players.[1] Evaluation functions often work by further abstracting the game-state to a class of states that are the all same along certain "feature" dimensions. A node in a graph of game-states can be viewed as an abstraction of all the possible game-histories in which that particular game-state corresponding to the node is reached.

However, we can imagine a number of additional kinds of abstraction that might be useful as well. We might want to abstract over states, sequences of states, and state-transitions, as suggested in the previous section. We can also draw ideas for kinds of abstraction from research in planning:

- We might want to think about several possible future moves/actions/state-transitions without specifying their chronological order. This kind of abstraction is the basis of nonlinear planning. It can also apply to game-playing: in a situation with many moves available, you might want to consider the possibility "do both (move A) and (move B)," without necessarily deciding, at first, which game-turn you will do the moves on, or even which one you will do first.
- We might want to be able to consider performing some large, high-level action, without specifying, at first, the atomic moves that will be performed in carrying out that action. For example, in chess we might want to think about the general consequences of attacking the opponent's pawn structure, before going to the trouble of figuring out exactly how, where, and when to do it. Such abstraction to high-level actions is the type of abstraction found in abstract planning.
- Very closely related to the above, is planning that works by coming up with relatively easy subgoals that might be useful in attaining our overall objective, and only later figures out exactly how to achieve them, and how to proceed after achieving them. For example, in chess, you might decide it would be good to capture some particular enemy piece, and only then figure out how. This kind of subgoaling is used by island-driven planning.

1. As well as, of course, the state of the rest of the world, outside the micro-world of the game.

4.4 A unifying view of abstraction

So given the large variety of kinds of abstraction found both in existing game-playing programs and in planners, it might be worthwhile to spend some time considering what they have in common, and whether a unified view of abstraction might offer some guidance in the quest to make game-playing programs more general and more capable.

At first the various kinds of abstraction we might want in a game-playing program seem very different. Might it be possible to express some of them in terms of others? The first kind of abstraction I mentioned was abstraction over some of the details of a world-state, for example, ignoring the identity of hidden cards in a card game. We can presumably form graphs of such abstract states, and describe possible transitions between them. For example, in a solitaire game, even if many cards are hidden, it may still possible to plan a sequence of several moves, as long as those moves don't depend on the identity of the cards that are still unknown.

But can the other kinds of abstractions be represented in terms of abstract states? For example, consider the nonlinear-planning type of abstraction, in which I consider doing move A and move B but I don't say when or in what order. The only way to coerce this into an abstract state is to have "a state sometime in the future in which move A and move B have been done" as a possible type of abstract state. However, this seems awkward and inelegant.

Instead, I take a different approach. Instead of building upwards from states, adding unnatural properties to them such as information about the history of previous states, I build top-down from abstract descriptions of possible entire histories of the world, adding restrictions to them, similarly to how Wellman defines abstract plans in [Wellman-88]. This is a simplified version of the framework discussed in the previous chapter.

Let us imagine there is some space Ω of all the possible complete histories of the world (or of the micro-world of the current game). Then let us define an *abstraction* A to be a description[1] of a class of world-histories. For any A there is some subset $S_A \subseteq \Omega$ containing the world histories that *satisfy* the description. An abstraction will generally describe some properties of the world-history, while leaving many others unspecified. Of

1. For a reference on the use of description taxonomies for knowledge representation, see [Woods-90].

course, to make this definition concrete we must have some well-defined language L in which we express our abstractions.

It is useful to associate with A the *abstraction-proposition* P_A that asserts that the actual world-history will turn out to satisfy A. An agent's belief or disbelief in P_A can be an important property of A.

All of the kinds of abstractions mentioned in section 4 can be subsumed under this definition, as follows:

- A node in a traditional graph of game-positions corresponds to an abstraction of the game-histories in which that game-position is reached.
- Likewise, a node in a traditional game-tree (with repeated states) corresponds to a proposition that the current game will begin with a particular sequence of moves (or game-positions).
- When a game-state is generalized to a class of similar states, for purposes of evaluation, the general class can be identified with a proposition that a game-state is reached that has such-and-such values for various features.
- We can abstract over hidden information or irrelevant details in the game-state with a proposition that a particular abstract (incompletely-specified) state is reached in the current game.
- We can represent abstract (long-term) actions with abstractions of world-histories in which the high-level action is performed.
- We can represent island subgoals with abstractions of world-histories in which the subgoal is achieved.
- We can represent partially-specified moves with propositions that a move satisfying certain conditions is made in the current game.

Moreover, this framework offers the possibility of being *less* abstract than the standard approach (depending on the language L). A proposition can include statements about the remaining time or about the present or future beliefs and intentions of the players; information that the limited concept of game position is not suited to express.

4.5 The development of abstract reasoning algorithms

Having a unifying conceptual framework is all well and good, but we still face the very hard problem of how to actually *do* effective, efficient reasoning involving abstractions. Two primary considerations affect my choice of how to approach this problem.

The first is the observation that existing game-playing programs and planning systems already do abstraction in several ways, as described above. By presenting a unifying definition for abstractions, we can more easily see how to take the existing methods that are being used for reasoning with these specific kinds of abstractions, and translate them to work on other kinds of abstraction as well. For example:

- Game-playing programs explore a partial graph of neighboring states, collect estimated value information at the fringe of the partial graph, and propagate this information back through the graph to make a decision at the root. We could do something similar for a partial graph of neighboring (closely-related) abstractions.
- Some planning systems start with abstract, partial plans and refine them to become more specific and more complete. We could do similar kinds of refinement in this more general framework of abstraction.

The second consideration is that probabilistic and decision-theoretic methods will be needed to interpret information gained by abstract reasoning, and allow efficient search in the relatively large and unconstrained space of abstractions. Decision theory will be needed even more than in regular state-space search, where it is being used already:

- Some game-playing programs (see [Baum-Smith-93], [Hansson-Mayer-90a]) fruitfully use probabilistic inference to learn evaluation functions that inductively evaluate states based on past experience with an abstract class of similar states. We want to apply these methods more generally, to use past experience with high-level abstractions to draw conclusions about lower-level abstractions that they subsume.
- Many researchers ([Baum-Smith-93], [Pearl-84]) have devised ways to treat infor-

mation gained from a partial game-tree search as probabilistic, and propagate this probabilistic information up the tree. We wish to use similar techniques to propagate information from more-specific abstractions to less-specific ones that subsume them.

- Some game-playing programs ([Baum-Smith-93], [Russell-Wefald-91]) successfully apply concepts from decision theory to control exploration of game-trees. We will also find it helpful to do decision-theoretic search control in the much larger and more difficult space of abstractions.

4.6 Probabilities and utilities of abstractions

Recent work in computer game-playing, such as [Baum-Smith-93] and [Russell-Wefald-91], has begun to use the decision-theoretic concept of expected utility to define the value of game-states and the usefulness of reasoning actions, and to use probabilities to express uncertainty about the results of future state-space search and about the future decisions that agents will make. Can we use the tools of decision theory in a similar way in reasoning about abstract future-histories, as opposed to specific game-states?

First, what would it mean for an abstraction, a general proposition about the future, to have a utility? If we assign a utility to a proposition about the future, what exactly does that mean, given that we don't know exactly how that abstraction will be instantiated?

There is a very similar problem in the mathematical logic of preference, where the issue is how to define what it means to prefer a proposition's being true to its being false. One approach, taken by Doyle et. al. in [Doyle-et-al-91], is to define preference among propositions in terms of preferences among the possible worlds that satisfy them.

Similarly, we might try to define the utility of a proposition in terms of utilities of possible worlds (or in our case, world-histories) that satisfy it. But how is this to be done? Different worlds satisfying the proposition might have wildly varying utilities, depending on exactly how the proposition is satisfied. For example, the utility of an abstraction in chess, such as capturing the enemy queen, might depend on what I do to achieve it, how much time I take to do it, and what will happen afterwards, all of which might not yet be specified.

A natural answer is that the utility of an abstract proposition is its typical utility if it were satisfied in a way we expect it to be satisfied. For example, I might expect that if I capture the queen I will probably damage my pawn structure in the process, that I will not spend much of my remaining clock time planning the capture, and that afterwards such-and-such other events are likely to occur.

In order to define this "typical" utility, we imagine that worlds have probabilities as well as utilities. Then the utility of an abstract proposition can be defined as the expected utility of the worlds that satisfy it. A side effect is that we can define the probability of the proposition as well.

More formally, let us imagine that at any given time, the agent α has a subjective probability distribution $\mathrm{Pr}_\alpha\{\omega\}$ over all the possible future world-histories $\omega \in \Omega$, and a subjective utility $\mathrm{Ut}_\alpha\{\omega\}$ for each of them as well. (This is for definitional purposes only; we don't require that our agents actually could assess probabilities and utilities for every possible world-history, just that the agents can be thought of as implicitly having such probabilities and utilities.)

Then, I define α's *probability* $\mathrm{Pr}_\alpha\{A\}$ for an abstraction $A \subseteq \Omega$ to be the sum of the probabilities of the world-histories in A:

$$\mathrm{Pr}_\alpha\{A\} = \sum_{\omega \in A} \mathrm{Pr}_\alpha\{\omega\} \tag{29}$$

And I define α's *utility* $\mathrm{Ut}_\alpha\{A\}$ for A to be α's mathematical expectation of the utility of the world-histories in A:

$$\mathrm{Ut}_\alpha\{A\} = \frac{\sum_{\omega \in A} \mathrm{Pr}_\alpha\{\omega\}\,\mathrm{Ut}_\alpha\{\omega\}}{\mathrm{Pr}_\alpha\{A\}}. \tag{30}$$

4.7 Abstraction networks

State-space-oriented game-playing programs generally work by examining a tree or graph of nodes representing game-states, connected by arcs representing legal state transitions. In order to translate state-space search algorithms into abstraction terms, we must first translate game-graphs. We define the translation to be a network of abstractions, connected by arcs representing various kinds of relationships between abstractions, the primary one being a subsumption relationship. Abstraction networks are thus similar to Wellman's plan-specialization graphs [Wellman-88], except that we allow other kinds of relationships between nodes besides subsumption, and nodes represent abstractions of any aspect of the world-history, not just abstractions over actions performed by the current agent.

As described in §4.4, a game-state s can be interpreted as an abstraction of the world-histories in which that game-state arises. A node in a typical game-tree, however, does not correspond exactly to a game-state, because there may be more than one node for each game-state (unless the concept of game-state is coerced to include the history of previous states). A node in a game-tree is, rather, an abstraction of the histories in which a certain *initial subsequence* $s_1, \ldots, s_n$ of states follows the current or root state. We presume that this kind of abstraction is describable in our abstraction language L, and we write this abstraction $\mathrm{seq}(s_1, \ldots, s_n)$. To complete the translation of game-trees, we need to translate the relation expressed by game-tree arcs into the abstraction framework. The normal parent-child relation in a game tree becomes a subsumption relationship between abstractions. (See figure 11.)

Note that in the abstraction net, the difference between the two occurrences of s_1 is made explicit, instead of being implicit in the graph structure.[1]

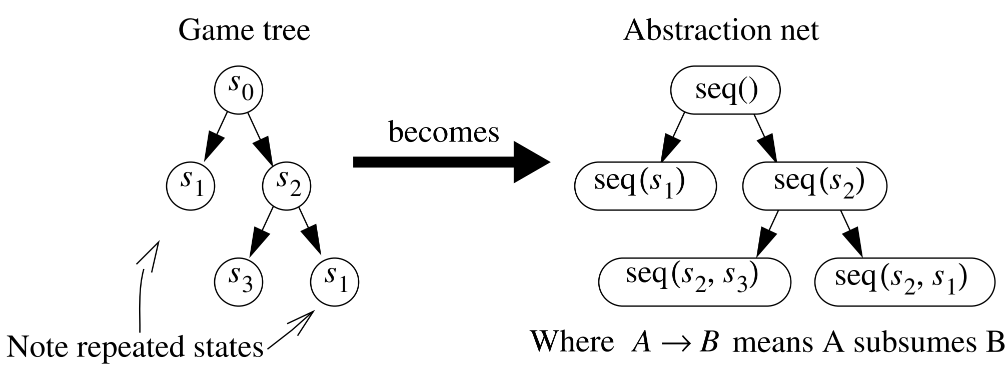


Figure 11. Translating a game tree into an abstraction network

1. For storage efficiency in an actual implementation, abstractions would be represented using references to other abstractions, reviving the dependence on structure. But that is an implementation issue; in a conceptual, graphical representation, explicitness is important for clarity of meaning.

We can also translate game graphs into abstraction nets, for the case of game-playing algorithms that can identify repeated states (with transposition tables, for example). However, space forbids describing this translation here.

4.8 Conclusion—state of work in progress

Now, having figured out what some existing techniques are doing in abstraction terms, I am working on developing algorithms for correctly propagating probability and utility information through abstraction nets containing arbitrary abstraction nodes connected by subsumption arcs and a few other types of relations. These algorithms are still in flux, so I will not attempt to describe their workings here. However, I will say that the methods I am investigating are partly inspired by the belief-network propagation algorithms described by Pearl in [Pearl-88], and by Baum and Smith's probabilistic game-tree methods [Baum-Smith-93], which we discussed in detail in chapter 2

Eventually, I plan to have the propagation algorithms specified well enough to implement them in a program to play Spider, thus improving on the straightforward decision-analysis approach to the game that we considered in the previous chapter. I plan to begin with support for only a few kinds of abstraction, and use hard-coded search control, and later, make the abstraction language broader and introduce decision-theoretic search control based on past experience.

Even after that is done, many issues still remain. My current approach is based on maximizing the expected value of the current game only, whereas a broader-thinking agent would also consider the utility of information gained in the current game for use in future games as well. Handling other agents better by reasoning about abstractions of their possible mental state is another crucial issue for more general and effective game-playing and planning. Finally, the ultimate challenge for planning, and AI in general, is to go beyond well-defined microworlds, and develop agents that can adapt to the complexity and ill-definedness of everyday life.

Chapter 5

Summary and Conclusions

In this thesis we have explored the development of decision-theoretic methods in AI, focusing on their development within computer game playing (CGP). We saw that the general theoretical concepts of probability and expected utility were vital for moving CGP techniques beyond the earlier *ad hoc*, ill-justified techniques, and towards more well-founded methods, that explicitly tried to approximate the normative recommendations of decision theory.

However, just as in other fields, when CGP researchers began to apply decision theory, they quickly encountered what might be considered its primary flaw: its lack of a direct recommendation for what to do when there is not sufficient time to compute the decision-theoretic optimal action. Russell and Wefald were spurred by this problem to begin investigations into a theory of rational metareasoning, which would hopefully provide a principled way to design agents despite the limited rationality problem.

Their research, and the even more sophisticated later techniques by Baum and Smith, inspired us to explore whether the techniques being developed for game-playing had finally reached a point of generality and sophistication such that they could be fruitfully reapplied to a more practical problem. We chose decision analysis as the general problem framework to investigate, because decision trees share many features in common with game trees. We wished to take the Shannon paradigm of growing and evaluating a partial tree that has been so successful in game-playing and see if, augmented with the new decision-theoretic tree growth techniques, it could be equally useful when performing the sort of very large decision-analyses that are done in medical and other domains.

We decided to begin our investigations with a more accessible domain than that of medical decision making, namely that of Spider solitaire. In contrast to two-player games, this game is such that a classical decision analysis may be performed on its game tree to choose an optimal move. Unfortunately, after some time spent considering how best to represent the moves and states in a Spider game to allow an efficient search, it was realized that most of the work in designing the program was going to be in finding the right way of working with the different levels of abstraction at which reasoning about Spider could be performed. The salient feature of decision-making in Spider was that it required exploring the right kinds of abstractions, not that it required delicate control of search within a fixed level of abstraction. The CGP search control techniques were essentially still inappropriate; Spider demanded a different kind of representation than the flat state-graph upon which all the CGP search control algorithms were based.

Although this was a disappointment with regard to the state of the art of limited rationality methods in decision-theoretic AI, it opened the door to a whole new area of investigation which is just now beginning to be explored. That area is the principled control of search through large, complex spaces of abstract states. The exciting possibility for future research is twofold: First, that the consideration of issues of abstraction, rather than an over-focus on concrete states, will allow the development of more general, robust, and powerful theoretical models for limited rationality (some investigations of which I have begun in chapter 3), and second, that in the meantime while such theories are being developed, the use of probabilistic and decision-theoretic techniques will aid greatly in the development of AI algorithms for reasoning with abstractions (which I have begun exploring in chapter 4). Other researchers, too, are already beginning to slowly explore limited rationality and abstraction techniques. Eventually there will be a symbiosis between LR ideas and abstraction ideas that will hasten the development of both into a common foundation for general reasoning.

As for computer game-playing, I believe that it has proven to be a fruitful breeding ground for many interesting ideas, but historically it has involved an overemphasis on particular games, and not enough of a focus on techniques designed to be readily generalizable to other domains. However, this problem can probably be fixed by exploring a greater variety of games beyond the classic chess-like games, such as games with much larger branching factors, such as Go ([Pell-90], [Brugmann-93]), and games with imperfect information, such as Solitaire and Bridge ([Gamback-et-al-91], [Smith-Nau-93]). Other ways we might try to improve CGP's relevance to AI are by concentrating on more general game-playing tasks such as Metagame [Pell-93], and by just resolving as individual researchers to focus more effort on developing generalizable methods, as Russell and Wefald and Baum and Smith have done.

In one sense, recent CGP research has been largely disjoint from AI, in that many of the AI people that were working on games eventually either turned into brute-force chess hackers, or left games because their more general methods could not compete with the discouragingly successful brute-force ones. But now, some AI people are coming back to games, as evidenced by the [Games-93] proceedings. However, as this happens, we should always keep in mind that although games are interesting and fun, a game-playing technique is worthless if it is not developed with an eye towards eventually generalizing it to work on problems of more general utility.

Although the research reported in this thesis has not so far achieved as much in terms of conclusive results as was originally hoped, we believe that this thesis still serves as a valuable contribution, surveying and tying together several lines of research, synthesizing a broad overall perspective on the situation in computer game-playing and decision-theoretic AI, and pointing out some very interesting and promising directions, namely limited rationality and abstract search, for further advances in decision-theoretic artificial intelligence.

Bibliography

[AAAI-80]

Artificial Intelligence: Proceedings of the 1st Annual National Conference, Stanford University, August 18-21, 1980. American Association for Artificial Intelligence.

[AAAI-87]

Artificial Intelligence: Proceedings of the 6th National Conference, Seattle, Washington, July 13-17, 1987. American Association for Artificial Intelligence. Morgan Kaufmann, Los Altos, California, 1987.

[AAAI-88]

Artificial Intelligence: Proceedings of the 7th National Conference, Saint Paul, Minnesota, August 21–26, 1988. American Association for Artificial Intelligence. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[AAAI-90]

Artificial Intelligence: Proceedings of the 8th National Conference, Boston, Massachusetts, July 29-August 3, 1990. American Association for Artificial Intelligence. AAAI Press / The MIT Press, Menlo Park, California.

[AAAI-91]

Artificial Intelligence: Proceedings of the 9th National Conference, Anaheim, California, July 14-19, 1991. American Association for Artificial Intelligence. AAAI/ MIT Press, Cambridge, Massachusetts.

[AAAI-92]

Artificial Intelligence: Proceedings of the 10th National Conference, San Jose, California, July 12-16, 1992. American Association for Artificial Intelligence. AAAI/ MIT Press, Cambridge, Massachusetts.

[Abramson-85]

Bruce Abramson. An explanation of and cure for minimax pathology. Technical Report UCLA-CSD-850034, University of California, Los Angeles, California, October 1985.

[AILR-89]

Michael Fehling and Stuart Russell, editors. *1989 Spring Symposium Series: AI and Limited Rationality: Working Notes*, Stanford University, March 28-30, 1989. American Association for Artificial Intelligence. Unpublished. Bibliography available on the World-Wide Web at URL `http://medg.lcs.mit.edu/mpf/papers/AAAI/AILR-89/AILR-89.html`.

[Albert-94]

David Z. Albert. Bohm's alternative to quantum mechanics. *Scientific American*, **270**(5):58-67, May 1994.

[Allis-et-al-91]

L. V. Allis, M. van der Meulen, and H. J. van den Herik. Proof-number search. Technical report 91-01, Department of Computer Science, University of Limburg, PO Box 616, 6200 MD Maastricht, The Netherlands. 1991.

[Anantharaman-et-al-90]

T. S. Anantharaman and M. S. Campbell and F.-H. Hsu. Singular extensions: Adding selectivity to brute-force searching. *Artificial Intelligence*, **43**(1):99–110, April 1990.

[Arthur-91]

W. Brian Arthur. Designing economic agents that act like human agents: A behavioral approach to bounded rationality. *Learning and Adaptive Economic Behavior*, **81**(2):353-359, May 1991.

[Bates-et-al-53]

M. Audrey Bates, B. V. Bowden, C. Strachey, and A. M. Turing. Digital computers applied to games. In [Bowden-53], chapter 25, pages 286-310.

[Baum-92]

Eric B. Baum. On optimal game tree propagation for imperfect players. In [AAAI-92], pages 507-512.

[Baum-93]

Eric B. Baum. How a Bayesian approaches games like chess. In [Games-93].

[Baum-Smith-93]

Eric B. Baum and Warren D. Smith. Best Play for Imperfect Players and game tree search. Draft report, available on the World-Wide Web at URL `ftp://external.nj.nec.com/pub/eric/game.ps.Z`, May 1993.

[Berlekamp-et-al-82]

Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*. Academic Press, New York, 1982. Two volumes.

[Blair-et-al-92]

Jean R. S. Blair, David Mutchler, and Cheng Liu. Heuristic search in one-player games with hidden information. Tech report, Computer Science Department, University of Tennessee at Knoxville. June 1992.

[Blair-et-al-93]

Jean R. S. Blair, David Mutchler, and Cheng Liu. Games with imperfect information. In [Games-93].

[Boddy-Dean-89]

Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In [AILR-89], pages 6–9.

[Bodkin-92]

Ronald J. Bodkin. Extending computational game theory: Simultaneity, multiple agents, chance and metareasoning. Master's thesis, Massachusetts Institute of Technology, September 1992.

[Bodkin-92a]

Ronald J. Bodkin. A corrected proof that SCOUT is asymptotically optimal. TR in preparation, 1992. Author's email: `<rjbodkin@medg.lcs.mit.edu>`.

[Bowden-53]

B. V. Bowden, editor. *Faster Than Thought: A Symposium on Digital Computing Machines*. Sir Isaac Pitman and Sons, Ltd., London, 1953.

[Breese-Horvitz-90]

John S. Breese and Eric J. Horvitz. Reformulating and solving belief networks under bounded resources. Technical report KSL-90-28, Knowledge Systems Laboratory, Stanford University, March 1990.

[Brugmann-93]

Bernd Brugmann. Monte Carlo Go. Technical report, Physics Department, Syracuse University. Author's email: `<bruegman@npac.syr.edu>`.

[Carmel-Markovitch-93]

David Carmel and Shaul Markovitch. Learning models of opponent's strategy in game playing. In [Games-93].

[Chi-Nau-87]

Ping-Ching Chi and Dana S. Nau. Comparing minimax and product in a variety of games. In [AAAI-87], pages 100-104.

[Dean-91]

Thomas Dean. Decision-theoretic control of inference for time-critical applications. *International Journal of Intelligent Systems*, **6**:417-441, 1991.

[Dean-Boddy-88]

Thomas Dean and Mark Boddy. An analysis of time dependent planning. In [AAAI-88], pages 49–54.

[Dean-et-al-92]

Thomas Dean, Jak Kirman, and Keiji Kanazawa. Probabilistic network representations of continuous-time stochastic processes for applications in planning and control. In *Proceedings of the First International Conference on AI Planning Systems*, 1992.

[Dean-et-al-93a]

Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Deliberation scheduling for time-critical sequential decision making. In [UAI-93], pages 309-316.

[Dean-et-al-93b]

Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning with deadlines in stochastic domains. In [AAAI-93], pages 574–579.

[Delcher-Kasif-92]

Arthur L. Delcher and Simon Kasif. Improved decision-making in game trees: Recovering from pathology. In [AAAI-92], pages 513-518.

[Doyle-et-al-91]

Jon Doyle, Yoav Shoham, and Michael P. Wellman. A logic of relative desire (preliminary report). In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems, 6*, volume 542 of *Lecture Notes in Artificial Intelligence*, pages 16-31. Springer-Verlag, Berlin, October 1991.

[Edwards-Hart-63]

Daniel J. Edwards and Timothy P. Hart. The α-β heuristic. Artificial Intelligence Project Memo 30, MIT Research Laboratory of Electronics and Computation Center, Cambridge, Massachusetts, October 28, 1963. Revised and renamed version of [Hart-Edwards-61].

[Feigenbaum-Feldman-63]

Edward A. Feigenbaum and Julian Feldman, editors. *Computers and Thought*. Robert E. Krieger Publishing Company, Inc., Malabar, Florida, reprint edition, 1981. Original edition 1963.

[Fuller-et-al-73]

S. H. Fuller, J. G. Gaschnig, and J. J. Gillogly. Analysis of the alpha-beta pruning algorithm. Technical report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, July 1973.

[Gamback-et-al-91]

Bjord Gamback, Manny Rayner, and Barney Pell. An architecture for a sophisticated mechanical Bridge player. In [Levy-Beal-91].

[Games-93]

Games: Planning and Learning: Papers from the 1993 Fall Symposium, Raleigh, North Carolina, October 22-24, 1993, American Association for Artificial Intelligence. AAAI Press, Menlo Park, California. Technical report FS9302.

[Ginsberg-Geddis-91]

Matthew L. Ginsberg and Donald F. Geddis. Is there any need for domain-dependent control information? In [AAAI-91].

[Good-62]

I. J. Good. Subjective probability as the measure of a non-measurable set. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Logic, Methodology and Philosophy of Science*, Stanford University Press, 1962. Reprinted in [Good-83].

[Good-83]

I. J. Good. *Good Thinking: The Foundations of Probability and Its Applications*, University of Minnesota Press, 1983.

[Gordon-93]

Steven Gordon. A comparison between probabilistic search and weighted heuristics in a game with incomplete information. In [Games-93].

[Gorry-et-al-73]

G. Anthony Gorry, Jerome P. Kassirer, Alvin Essig, and William B. Schwartz. Decision analysis as the basis for computer-aided management of acute renal failure. *The American Journal of Medicine*, **55**:473–484, October 1973.

[Hacking-67]

Ian Hacking. Slightly more realistic personal probability. *Philosophy of Science*, **34**:311-325, December 1967. Revised from original version presented at the Western Division of the A.P.A. symposium, Chicago, May 4-6, 1967.

[Hamming-80]

Richard W. Hamming. *Coding and Information Theory.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1980.

[Hansson-Mayer-89a]

Othar Hansson and Andrew Mayer. Decision-theoretic control of search in BPS. In [AILR-89], pages 59-63.

[Hansson-Mayer-89]

Othar Hansson and Andrew Mayer. Heuristic search as evidential reasoning. In [UAI-89].

[Hansson-Mayer-90a]

Othar Hansson and Andrew Mayer. Probabilistic heuristic estimates. *Annals of Mathematics and Artificial Intelligence*, **2**:209-220, 1990.

[Hart-Edwards-61]

Timothy P. Hart and Daniel J. Edwards. The Tree Prune (TP) algorithm. Artificial Intelligence Project Memo 30, MIT Research Laboratory of Electronics and Computation Center, Cambridge, Massachusetts, December 4, 1961. Later revised and renamed to become [Edwards-Hart-63].

[Hart-et-al-68]

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on SSC*, **4**:100–107, 1968.

[Hart-et-al-72]

P. E. Hart, N. J. Nilsson, and B. Raphael. Correction to ‘a formal basis for the heuristic determination of minimum cost paths’, *ACM SIGART Newsletter*, **37**:28–29.

[Heckerman-et-al-89]
David E. Heckerman, Eric J. Horvitz, and Bharat N. Nathwani. Toward effective normative decision systems: Update on the Pathfinder project. Tech report KSL-89-25, Knowledge Systems Laboratory, Stanford University, March 1989.

[Hintikka-75]
Jakko Hintikka. Impossible possible worlds vindicated. *Journal of Philosophical Logic*, **4**:475-484, 1975.

[Hogenhuis-et-al-92]
W. Hogenhuis, S. K. Stevens, P. Wang, J. B. Wong, N. A. M. Estes, A. Manolis, and S. G. Pauker. Influence of quality of life on the cost-effectiveness of catheter ablation in patients with asymptomatic WPW syndrome. *Medical Decision Making*, **12**:340, 1992. Presented at the Society for Medical Decision Making, 13th Annual Meeting, Portland, Oregon.

[Honkapohja-93]
Seppo Honkapohja. Adaptive learning and bounded rationality. *European Economic Review*, **37**:587-594, 1993.

[Horvitz-88]
Eric J. Horvitz, John S. Breese, and Max Henrion. Decision theory in expert systems and artificial intelligence. *Journal of Approximate Reasoning*, special issue on uncertain reasoning, **2**:247–302, July 1988.

[Horvitz-88a]
Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In [AAAI-88], pages 111–116.

[Horvitz-89]
Eric J. Horvitz. Rational metareasoning and compilation for optimizing decisions under bounded resources. In [CI-89].

[Horvitz-Breese-88]
Eric J. Horvitz and John S. Breese. Ideal partition of resources for metareasoning. Tech report KSL-88-13, Knowledge Systems Laboratory, Stanford University, Stanford, California, 1988.

[Horvitz-et-al-89a]
Eric J. Horvitz, Gregory F. Cooper, and David E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In [IJCAI-89], pages 1121–1127.

[Horvitz-et-al-89b]
Eric J. Horvitz, David E. Heckerman, Keung-Chi Ng, and Bharat N. Nathwani. Heuristic abstraction in the decision-theoretic Pathfinder system. In [SCAMC-89], pages 178–182.

[Horvitz-et-al-89c]
Eric J. Horvitz, H. Jacques Suermondt, and Gregory F. Cooper. Bounded conditioning: flexible inference for decisions under scarce resources. In [UAI-89], pages 182-193.

[Hsu-et-al-90]
F.-H. Hsu and T. S. Anantharaman and M. S. Campbell and A. Nowatzyk. Deep thought. In [Marsland-Schaeffer-90], pages 55–78.

[IJCAI-81]
Artificial Intelligence: Proceedings of the 7th International Joint Conference, Vancouver, B.C., Canada, August 24-28, 1981. Morgan Kaufmann, Inc., Los Altos, California.

[IJCAI-89]
Artificial Intelligence: Proceedings of the 11th International Joint Conference, Detroit, Michigan, August 20-25, 1989. Morgan Kaufmann, San Mateo, California.

[IJCAI-93]
Artificial intelligence: Proceedings of the 13th International Joint Conference, Chambery, France, August 28-September 3, 1993. Morgan Kaufmann, San Mateo, California.

[Knoblock-91]
Craig Alan Knoblock. Automatically generating abstractions for problem solving. Technical Report CMU-CS-91-120, Carnegie Mellon University, School of Computer Science, Pittsburgh, May 1991. Ph.D. thesis.

[Knuth-Moore-75]
Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, **6**:293-326, 1975.

[Konolige-86]
Kurt Konolige. What awareness isn't: A sentential view of implicit and explicit belief. In Joseph Y. Halpern, editor, *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 241-250, Los Altos, California, 1986. Morgan Kaufmann.

[Levy-Beal-91]
D. N. L. Levy and D. F. Beal, editors. *Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad*. Ellis Horwood, 1991.

[Lilly-94]
Gregory Lilly. Bounded rationality: A Simon-like explication. *Journal of Economic Dynamics and Control*, **18**:205-230, 1994.

[Lipman-89]
Barton L. Lipman. How to decide how to decide how to ... : Limited rationality in decisions and games. Working paper, Carnegie Mellon University, Pittsburgh, Pennsylvania, February 1989. Also appears in [AILR-89].

[Lipman-91]

Barton L. Lipman. How to decide how to decide how to ... : Modeling limited rationality. *Econometrica*, **59**(4):1105-1125, July 1991.

[Marsland-Schaeffer-90]

T. Anthony Marsland and Jonathan Schaeffer. *Computers, Chess, and Cognition*. Springer-Verlag, 1990.

[McCarthy-90]

John McCarthy. *Chess as the Drosophila of AI*. Chapter 14 of [Marsland-Schaeffer-90], pages 227–237.

[McNeil-et-al-81]

Barbara J. McNeil, Ralph Weichselbaum, and Stephen G. Pauker. Speech and survival: Tradeoffs between quality and quantity of life in laryngeal cancer. *New England Journal of Medicine*, **305**:982-987, 1981.

[Morehead-68]

Albert H. Morehead, editor. *Official Rules of Card Games*, Fawcett Crest Books, New York, 1968.

[Moskowitz-84]

A. J. Moskowitz, V. H. Dunn, J. Lau, and Stephen G. Pauker. Can "hypersimplified" decision trees be used instead of Markov models? *Medical Decision Making*, **4**(4), 1984.

[Nau-80]

Dana S. Nau. Pathology on game trees: A summary of results. In [AAAI-80], pages 102-104.

[Nau-82]

Dana S. Nau. An investigation of the causes of pathology in games. *Artificial Intelligence*, **19**(3):257-278, 1982.

[Newborn-89]

Monroe Newborn. Computer chess: Ten years of significant progress. *Advances in Computers*, **29**:197-248, 1989.

[Newell-et-al-58]

Allen Newell, J. C. Shaw, and H. A. Simon. Chess-playing programs and the problem of complexity. *IBM Journal of Research and Development*, **2**:320-335, October 1958. Reprinted in [Feigenbaum-Feldman-63], pages 39-70.

[Norman-Shimer-94]

Alfred L. Norman and David W. Shimer. Risk, uncertainty, and complexity. *Journal of Economic Dynamics and Control*, **18**:231-249, 1994.

[Pauker-92]
S. G. Pauker and M. H. Eckman. Decision theory. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, 2nd edition, John Wiley and Sons, New York, 1992, pages 317–320.

[Pearl-81]
Judea Pearl. Heuristic search theory: Survey of recent results. In [IJCAI-81], pages 554-562.

[Pearl-83]
Judea Pearl. On the nature of pathology in game searching. *Artificial Intelligence*, **20**:427-453, 1983.

[Pearl-84]
Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley series in Artificial Intelligence. Addison-Wesley, 1984.

[Pearl-88]
Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

[Pearlman-Jonsen-85]
R. A. Pearlman and A. Jonsen. The use of quality-of-life considerations in medical decision making. *Journal of the American Geriatric Society*, **33**(5):344–352, May 1985.

[Pell-90]
Barney Pell. Exploratory learning in the game of Go: Initial results. In [Levy-Beal-91]. Also appears as University of Cambridge Computer Laboratory Technical Report No. 275.

[Pell-93]
Barney Pell. Strategy Generation and Evaluation for Meta-Game Playing. Ph.D. thesis, Computer Laboratory, University of Cambridge, 1993. Also appears as University of Cambridge Computer Laboratory Technical Report No. 315.

[Pliskin-et-al-80]
J. S. Pliskin and D. S. Shepard and M. C. Weinstein. Utility functions for life years and health status. *Operations Research*, **28**:206–224, 1980.

[Prinz-52]
D. G. Prinz. Robot chess. *Research*, **5**:261-266, 1952.

[Rich-Knight-91]
Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, Inc., 2nd edition, 1991.

[Rivest-88]
R. L. Rivest. Game tree searching by Min/Max Approximation. *Artificial Intelligence*, **34**:77–96, 1988.

[Rivest-Sloan-93]
Ronald L. Rivest and Robert H. Sloan. On choosing between experimenting and thinking when learning. *Information and Computation*, **106**(1):1–25, September 1993.

[Russell-90]
Stuart Russell. Fine-grained decision-theoretic search control. In [UAI-90], pages 436-442.

[Russell-Wefald-89]
Stuart Russell and Eric Wefald. On optimal game-tree search using rational meta-reasoning. In [IJCAI-89], pages 334-340.

[Russell-Wefald-91]
Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, **49**:361-395, 1991. Elsevier Science Publishers.

[Russell-Wefald-91a]
Stuart Russell and Eric Wefald. *Do the Right Thing: Studies in Limited Rationality.* The MIT Press, Cambridge, Massachusetts, 1991. Artificial Intelligence series.

[Russell-et-al-93]
Stuart J. Russell, Devika Subramanian, and Ronald Parr. Provably bounded optimal agents. In [IJCAI-93], pages 338-344.

[Sacerdoti-74]
Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, **5**:115–135, 1974.

[Samuel-59]
A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**:211-229, July 1959. Reprinted in [Feigenbaum-Feldman-63], pages 71-105, and in [Shavlik-Dietterich-90], pages 535-554.

[Savage-67]
L. Savage. Difficulties in the theory of personal probability. *Philosophy of Science*, **34**:305-310, December 1967.

[Schaeffer-91]
J. Schaeffer. Conspiracy numbers. *Artificial Intelligence*, **43**:67–84, 1990.

[Schwartz-et-al-73]
William B. Schwartz, G. Anthony Gorry, Jerome P. Kassirer, and Alvin Essig. Decision analysis and clinical judgement. *The American Journal of Medicine*, **55**:459–472, October 1973.

[Shannon-50]
Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine* (seventh series), **XLI**(314):256-275, March 1950. First presented at the National Institute of Radio Engineers Convention, March 9, 1949, New York.

[Shavlik-Dietterich-90]
Jude W. Shavlik and Thomas G. Dietterich, editors. *Readings in Machine Learning*. Morgan Kaufmann, 1990.

[Simon-55]
Herbert A. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, **69**:99-118, 1955.

[Simon-82]
H. A. Simon. *Models of Bounded Rationality*, MIT Press, Cambridge, MA.

[Slagle-Dixon-69]
James R. Slagle and John K. Dixon. Experiments with some programs that search game trees. *Journal of the Association for Computing Machinery*, **16**(2):189-207, April 1969.

[Smith-92]
Warren D. Smith. Approximation of staircases by staircases. Technical report 92-109-3-0058-8, NEC Research Institute, Inc., 4 Independence Way, Princeton, New Jersey, December 1992.

[Smith-Nau-93]
Stephen J. Smith and Dana S. Nau. Strategic planning for imperfect-information games. In [Games-93].

[Smith-Nau-93a]
Stephen J. J. Smith and Dana S. Nau. Toward an analysis of forward pruning. In [Games-93].

[Sonnenberg-Beck-93]
Frank A. Sonnenberg and J. Robert Beck. Markov models in medical decision making: A practical guide. In *Medical Decision Making*, Hanley and Belfus, Inc., Philadelphia, Pennsylvania, 1993.

[Tietz-92]
Reinhard Tietz. Semi-normative theories based on bounded rationality. *Journal of Economic Psychology*, **13**:297-314, 1992.

[Truscott-81]
T. R. Truscott. Techniques used in minimax game-playing programs. Masters thesis, Duke University, Durham, North Carolina, 1981.

[Tsevat-et-al-88]
J. Tsevat, M. H. Eckman, R. A. McNutt and S. G. Pauker. Quality of life considerations in anticoagulant therapy for patients with dilated cardiomyopathy. *Medical Decision Making*, **8**:342, 1988. Presented at the Society for Medical Decision Making, Tenth Annual Meeting, Richmond, Virginia, October 17–19, 1988.

[Tversky-Kahneman-87]
A. Tversky and D. Kahneman. Rational choice and the framing of decisions. In R. Hogarth and M. Reder, editors, *Rational Choice: The Contrast between Economics and Psychology*, University of Chicago Press, Chicago, 1987.

[UAI-89]
Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence, Windsor, Ontario, August 1989.

[UAI-90]
Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, Cambridge, Massachusetts, July 1990.

[vanLent-Mutchler-93]
Michael van Lent and David Mutchler. A pruning algorithm for imperfect information games. In [Games-93].

[vonNeumann-28]
John von Neumann. Zur theorie der gesellschaftsspiele. *Math. Annalen*, **100**:295-320, 1928.

[vonNeumann-Morganstern-44]
John von Neumann and Oskar Morganstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, 1944. Later editions published by John Wiley and Sons, Inc.

[Wall-93]
Kent D. Wall. A model of decision making under bounded rationality. *Journal of Economic Behavior and Organization*, **21**:331-352, 1993.

[Wellman-88]
Michael P. Wellman. Formulation of tradeoffs in planning under uncertainty. Technical Report MIT-LCS-TR-427, Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, Massachusetts, August 1988. Ph.D. thesis.

[Winston-92]
Patrick Henry Winston. *Artificial Intelligence*, 3rd edition. Addison-Wesley, 1992.

[Winter-75]
S. Winter. Optimization and evolution in the theory of the firm. In R. H. Day and T. Groves, editors, *Adaptive Economic Models*, pages 73-118. Academic Press, New York, 1975.

[Wong-et-al-90]
John B. Wong, Frank A. Sonnenberg, Deeb N. Salem and Stephen G. Pauker. Myocardial revascularization for chronic stable angina: Analysis of the role of percutaneous transluminal coronary angioplasty based on data available in 1989. *Annals of Internal Medicine*, **113**(11):852–871, December 1990.

[Woods-89]
Donald R. Woods, in the introduction to the on-line help for the `spider` program, a freely-distributable UNIX program available on the Internet.

[Woods-90]
William A. Woods. Understanding subsumption and taxonomy: A framework for progress. Technical report 19-90, Center for Research in Computing Technology, Harvard University, 1990. Also appears in *Principles of Semantic Networks*, edited by John F. Sowa.

[Zilberstein-93]
Shlomo Zilberstein. Operational rationality through compilation of anytime algorithms. Ph.D. dissertation UCB/CSD 93/743, Computer Science Division (EECS), University of California, Berkeley, California, May 1993.

[Zilberstein-Russell-93]
Shlomo Zilberstein and Stuart J. Russell. Anytime sensing, planning and action: A practical model for robot control. In [IJCAI-93], pages 1402-1407.