Rational partial state-graph search

Michael P. Frank
MIT Lab for Computer Science
Email: mpf@medg.lcs.mit.edu

Abstract.

The AI technique of searching through trees or graphs of neighboring states in a *state-space* has been used for many years in game-playing and problem-solving domains. For many of these domains, the state-spaces are too large to search completely; there is only time for a *partial* state-space search. Any partial search technique must address two issues: what part of the space do we search, and how do we obtain useful information from a partially-searched space?

Recently, a number of research groups have developed *rational* techniques, by which I mean ones based on decision theoretic principles, that answer one or both of these questions. These researchers have argued convincingly that their rational approaches improve on the non-decision-theoretic techniques previously used for partial state-space search.

In this paper, I will survey work in this area by researchers such Hansson & Mayer, Russell & Wefald, and Baum & Smith, with an emphasis on the latter, since it subsumes much of the earlier work. I will conclude with some speculations on how to apply the principles learned from rational state-space search to do abstract reasoning that transcends the state-space paradigm.

Contents.

1 Introduction

People often simplify problems in order to solve them. However, in AI we need to simplify more than most people, because it's our not-so-bright computer programs who must do the solving. One classic AI simplification of a problem domain is to characterize it in terms of a well defined set of possible states and state-transitions. For those problems in which this simplification is reasonably appropriate, it helps us a lot by giving our programs a nice, well-defined 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.

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? 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 using information about part of the space to jump to conclusions about the space as a whole.

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, they 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 action, finally, to do.

Progress is being made steadily along these lines. In my opinion, the current pinnacle of this work is the recent draft paper by Eric Baum and Warren Smith of NEC Research Institute. They describe, in fairly thorough detail, a complicated but impressively well-justified set of methods for doing partial state-space search in a decision-theoretic way. The main goal of this paper is to summarize and explain these methods. A subsidiary goal, partly inspired by Baum and Smith's work, is to begin to examine how some of these methods might be generalized to guide reasoning even if some of the simplifying assumptions implicit in the classical state-space paradigm are removed.

1.1 Structure of this paper .

Section 2 attempts to remind the reader of some of the assumptions implicit in the traditional paradigm of partial state-space search, and can be skipped without much impact on the rest of the paper. Section 3 briefly reviews the earlier work in the field leading up to Baum and Smith. Section 4 describes Baum and Smith's approach in detail. Section 5 speculates on further work that generalizes their methods. Section 7 concludes. Appendix A is a proposal for a specific improvement of part of Baum and Smith's technique.

2 Partial state-graph search

This section is intended primarily for novice readers not familiar with the AI technique of searching through state-space graphs. However, it may also be of interest to those AI researchers who enjoy reading surveys that attempt to summarize and unify the sometimes-disparate work in the field. The following subsections provide some philosophical background for my treatment of the techniques described in the rest of the paper, and provide a sort of unifying synthesis of much of what has been going in AI with regard to both game-playing and puzzle-solving.

2.1 State-graph ontology

A typical AI program performs some sort of reasoning concerning some application domain that is of interest to the program's author. An important characteristic of any agent is the ontological stance it exhibits towards the domain of application, i.e., its model of the domain.

A very common stance taken by AI programs towards possible circumstances in their domains is to think of these circumstances in terms of *concrete microworld states*, that is, complete assignments of values to a certain set of properties that are relevant in whatever *microworld* (some small piece of the world) that the program has been built to reason about. Concrete microworld states are usually just referred to in AI as *world-states,* or just *states,* but I give them the longer name to clearly identify what they are. For now, however, I will just write *states.*

In the stance taken by such programs, the microworld is thought of as being in a definite state at any given time; time itself is often discretized to a sequence of relevant moments, with the world changing from one state to another between moments. There is often a notion of a set of the legal or possible or consistent states that the world could ever be in; certain states are thought of as possible circumstances, while others are not. This set is referred to as the microworld's *state-space.*

Additionally, these programs typically have a concept of possible change, expressed in terms of primitive *state transitions* (or "operators" or "actions") that are imagined to transform the world from one state to another as time passes; there is imagined to be a set of allowed 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.

In particular, the vast majority of the AI programs written for game-playing and puzzle-solving domains have adopted the state graph, or its specialization the *state tree*, as their primary conceptualization of their domain's structure. This has proved to be a natural and fruitful framework in which to develop well-defined and well-understood algorithms for reasoning in such domains. This success is no accident, since these domains are artificial ones that were defined by humans in terms of well-defined states and state-transitions in the first place. Humans make use of the state-graph conceptualization when doing analysis of these domains.

However, I will argue in §5 that, even in 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 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/puzzle domains.

2.2 Partial state-graph epistemology

Another important characteristic of any agent is the scope of its knowledge about a domain it is working in. In particular, if we're 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 particular possible circumstances that might occur in that domain.

Given the state-graph ontology described above, 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 the set of nodes and arcs that the agent has *examined*. A node or arc is said to have 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 have knowledge about nodes that it has not examined; for example, it may have proven that all nodes in an entire unexamined region of the graph have a certain property. Nevertheless, we usually still consider the examined partial graph to be a special component of an agent's knowledge, because there is usually a lot of information about states that the agent can only obtain (as far as we know) by examining the corresponding part of the state-graph.

For some small game and puzzle domains (e.g., tic-tac-toe, the 8-puzzle) the state-graph is so simple so that a program can quickly examine the entire microworld state-graph. In these domains, we can obtain and process all the information we need on which to base a solution/action/decision/answer very quickly, and there is not much to say about epistemology.

More interesting are 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 state-graph epistemology of our agents becomes an important factor in deciding how they ought to go about their reasoning. The examined partial graph cannot be the full graph, so the knowledge that will available to our agents at a time when they must make a decision will depend strongly on how the agent reasons.

2.3 Partial state-graph methodology

Thus we are led to the two crucial methodological questions for agents having a partial state-graph epistemology:

1. What partial state-graph shall I examine?
2. How can I derive useful knowledge by examining only a partial graph?

I submit that the way a program answers the above two questions is a useful characteristic for categorizing AI programs whose knowledge is based on partial state-graphs. I believe that the work done by most of these programs can be naturally broken down into computation directed towards 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 is as opposed to 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* that can be stopped at any time and asked to immediately provide an answer based on the partial graph examined so far (such as in iterative deepening). Another reason 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 nodes and only keeping the useful information that is derived from them (such as in alpha-beta). Note that expunged nodes are still considered part of the examined partial graph, in my epistemology.

Another characteristic of most partial-state-graph-search programs is that the examined partial graph is usually connected. The connectedness is the consequence of the natural graph-examination scheme of always examining nodes that are connected to ones already examined. The connectedness property also tends to make it easier to extract useful information from the partial graph. However, one can imagine a partial graph-search scheme that examines a scattering of randomly-chosen nodes that are not inaccessibly connected to each other, or that expands the graph outwards from a small selection of unconnected search “islands.”

2.3.1 Choosing a partial graph to examine

The first fundamental 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* is a single-agent example of an incremental search.

2.3.2 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 for this is to obtain some information at the nodes at the edges of the graph (leaves, if a tree), and propagate that information to the current state along the arcs of the graph, transforming the information as it goes, to determine its relevance to the overall decision or problem faced in the current state.

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 hand-made by a human.

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

Certain of the propagation algorithms implictly assume that leaf information correctly represents a property of the entire unexplored part of the graph below that leaf. The rational approaches attempt to weaken this assumption.

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 of a DAG. The reason for assumption is that the correct algorithm is too slow or too hard to describe. Such assumptions may even be used in cases when they are known to be false; the hope is 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 characterize.

3 Earlier approaches

Non-rational methods for partial graph (often tree) construction and evaluation were used for many years. The early non-rational methods were things like minimax in 2-player games, and A* in single-agent search.

Minimax assumes the correctness of some heuristic function for evaluating leaves of a tree, and it assumes perfect play by both players, and then given these assumptions, it correctly propagates those values to make a decision at the root. Minimax searches a fixed part of the space. One useful feature of minimax is that the amount of time it takes can be easily limited.

A*, on the other hand, depends on a heuristic function for search control, but makes no assumptions about the value of positions other than the solution. A* always searches until it really does find a solution, and if the heuristic isn't extremely good, that can take a long time. Thus A* might be a poor algorithm if time is important.

There have been various tweaks to these sorts of algorithms to improve their performance, such as alpha-beta search with iterative deepening. There are also many similar alternative algorithms, such as Pearl's SCOUT [3]. However, in game playing, there was, for a long time nothing that did any sort of selective search control.

Then ad-hoc methods for selective game-search, such as singular extensions, were developed. David McAllester [5] proposed a more unified, general, well-justified alternative to these methods, called conspiracy search, which roughly measured which parts of the search tree, if explored more deeply, would be most likely to have an important effect on our current situation. McAllester's method wasn't explicitly founded on probabilities or on the decision-theoretic principle of maximizing expecting utility, but it was similar in spirit; other researchers later reinterpreted McAllester's algorithm in terms of how well it approximated the decision-theoretic ideal.

Meanwhile, other researchers were looking at ways to define tree evaluation in ways that didn't make minimax's assumption of correct heuristic leaf values. The dominant method for handling uncertainty about the true value of nodes has been in terms of probability. Berliner proposed the B* algorithm [7], which worked with bounds on the probability of winning, rather than definite values. Pearl described in [3] how to use entire probability distributions over the space of game outcomes, under the assumption that distributions at different leaves are independent. However, like minimax, these distributions were over what the node value would be if we assumed perfect play by other players.

Other Hansson and Andrew Mayer [6] described a system called BPS (Bayesian problem solver) in which an ordinary evaluation function on nodes could be considered as evidence, in the Bayesian sense, of the node’s true value; the probability distributions over nodes’ values can then propagated through a tree like in a Bayesian belief network. An interesting characteristic of Hansson & Mayer’s technique was that it used the heuristic values of interior nodes as well as those of leaf nodes; whether that might also make sense in the context of other tree-evaluation algorithms remains unclear. BPS is a well-justified way to use heuristic node values, but it says nothing about how to grow a good tree.

3.0.1 Russell & Wefald’s MGSS*

Finally, a method for rational game-tree growth was described by Stuart Russell and Eric Wefald and implemented in their program MGSS* [2]. 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.

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 our overall, external decision. I.e., computational utility should be tied to real-world utility. They pinned down the theory of this idea fairly well. Another fundamental insight was that to do this, we need nodes to have distributions that describe what our estimate of the node’s value will be in the future, not Pearl’s distributions over what the true value of the node is.

A third basic contribution of Russell & Wefald’s work was to define the first-ever principled criteria 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.

MGSS* embodied 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. A more fundamental problem 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 calls “conspiracy depth 1.” Also, the asymptotic running time of their algorithm was a superlinear function of the size of the tree produced. Finally, after growing the tree, MGSS* just did an ordinary minimax on that tree, making the old assumption of the correctness of leaf values.

However, despite these drawbacks, Russell & Wefald demonstrated that their program performed much better than alpha-beta for a given number of nodes searched. (Note that this does not imply that it performed better for a given amount of computation time.)

Thus MGSS* paved the way for Baum and Smith’s work, which contains all the advantages of Russell and Wefald’s approach, and none of the above-mentioned problems.

4 Baum-Smith search

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.

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.

Section 4.1 describes Baum-Smith tree evaluation, while §4.2 describes tree growth.

4.1 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 (negamax) 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.

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" (§4.1.1). From this is derived a recursive strategy for node valuation called "Best Play for Imperfect Players," or BPIP (§4.1.2). 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" (§4.1.3). DFISA, in turn,

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.

requires an evaluation function that returns good probability distributions; Baum and Smith describe a statistics-gathering technique for generating such. (§4.1.4). Finally, information gathered at the leaves must be propagated up the tree; a few simple operations are described for doing this (§4.1.5).

4.1.1 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:

$$\Phi(a) = \wp\{A=a\} = \prod_\lambda \Phi_\lambda(\upsilon_a(\lambda)), \qquad \text{(EQ 1)}$$

where λ ranges over all the leaves of T, and $\upsilon_a(\lambda)$ 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:

$$\Phi_\nu(x) = \sum_{a:\ (\upsilon_a(\nu) = x)} \Phi(a), \qquad \text{(EQ 2)}$$

where $\upsilon_a(\nu)$ is defined as the game-theoretic (negamax) 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 expressed in terms that don't require large constructions like the ensemble.

4.1.2 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

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 (cite Ron).

expectation of the value[1] of the final outcome of the interaction, given only a partial *game-tree* of reachable states. The BPIP strategy assumes, as negamax does, that both players are using the strategy and that this is common knowledge. However, BPIP abandons negamax'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.[2]
- 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.
- $\wp_\alpha\{P\}$ 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:

$$E_\alpha[X] = \sum_x x \wp_\alpha\{X{=}x\} . \quad \text{(EQ 3)}$$

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

$$\upsilon_{\alpha_T}(\nu) = E_{\alpha_T}[\Upsilon_\nu] , \quad \text{(EQ 4)}$$

  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).

1. 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.

2. We assume that there is a definite agent whose turn it is at each node. For chance nodes, "nature" is treated as an agent.

- 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:

$$\upsilon_\alpha(\nu) = E_\alpha[\upsilon_\alpha(\beta_{\underline{\alpha}}(\nu))] = \sum_i \upsilon_\alpha(\nu_i)\wp_\alpha(\beta_{\underline{\alpha}}(\nu)=\nu_i) \qquad \text{(EQ 5)}$$

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]

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 EQ 5 simplifies to:

$$\upsilon_\alpha(\nu) = \upsilon_\alpha(\beta_\alpha(\nu)). \qquad \text{(EQ 6)}$$

To sum up EQ 5, 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. EQ 6 says that the value of the root is the value of its best child.

Now, the key difference between BPIP and negamax is that BPIP doesn't 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,

1. α cannot 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 $\beta_{\underline{\alpha}}$.

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

So, a BPIP agent is assumed, unlike negamax, 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 negamax, 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 [EQ 5]. 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.

4.1.3 Depth-free independent staircase approximation

We have seen that BPIP doesn't 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 EQ 5 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 §4.1.1. (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 §4.1.5.

4.1.4 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, that 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 negamax 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 [9].

4.1.5 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:

$$\bar{c}^{(\nu)}(x) = \sum_{i : x_i^{(\nu)} \geq x} p_i^{(\nu)} \qquad \text{(EQ 7)}$$

and

$$\underline{c}^{(\nu)}(x) = \sum_{i : x_i^{(\nu)} \leq x} p_i^{(\nu)} \qquad \text{(EQ 8)}$$

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 EQ 5 of computing probabilities of moves.

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

$$\underline{c}^{(\nu)}(x) = \prod_i \underline{c}^{(\nu_i)}(x). \qquad \text{(EQ 9)}$$

The formula for the upper CDF is similar. If we wish to have a negamax (as opposed to minimax) representation of value, we can substitute $\bar{c}$ for $\underline{c}$ and $-x$ for x on the right side of EQ 9.

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

$$\underline{c}^{(\nu)}(x) = \sum_i \wp_i^{(\nu)} \underline{c}^{(\nu_i)}(x), \tag{EQ 10}$$

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 EQ 9 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.

4.2 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 re-evaluation. 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 §4.2.1.

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 disprove 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 §4.2.5.

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're 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'd 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 §4.2.4.

4.2.1 Gulp trick

The "gulp trick" is a cost-amatorization 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 re-evaluating the tree after each growth step is amatorized 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 *slowdown 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 re-evaluate 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.

4.2.2 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 because of the limitation to conspiracy-depth-1 trees, and for other reasons which I won't go into here.

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.

(However, Baum and Smith do not actually characterize ESS in terms of U_continue.)

4.2.3 Expected Step Size

[This section needs to be rewritten to explain ESS in more detail and mention the ESS approximation theorem.]

The "Expected step size," or ESS, of a leaf is defined to be the expected absolute value of the change in [???] 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 §4.2, 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 attempt to justify ESS in these terms, and instead put forth a number of less satisfying arguments of why ESS is a good measure. [I would subjectively judge ESS to be the weakest link in Baum and Smith's technique, and the part most in need of empirical justification. (remove?)]

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.

4.2.4 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 (EQ 9 and EQ 10) 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.

1. As mentioned in §4.2.2, I interpret EUAFS as an attempt to approximate U_{continue}, the utility of continuing to search instead of stopping after this round of expansion.

4.2.5 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, 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 (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's very little to 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, doesn't address this kind of reasoning at all. I will discuss the issue of time some more in §5.

4.3 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 hand-made 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 aren't 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 (§4.1.3).
- Gulp trick (§4.2.1).
- Additive leaf expansion importance (§4.2.2).
- Additive time cost (§4.2.5).

Additionally, Baum and Smith's method is currently not very approachable; parts of their description are 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 the current paper will, as a supplement to the Baum and Smith draft paper, serve to clarify some of their ideas.

5 What next?

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 in §2), and determine which are most necessary for future progress, and which would be beneficial to try working without. Sections §5.1 and §5.2 reexamine a few of these assumptions and suggest how they might be relaxed. Sections §5.3 and §5.4 examine some unrelated methodological issues.

5.1 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[4]. 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.

5.2 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). 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 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 games 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:

1. 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 what I will do after revealing some hidden card, without having to repeat my work for every possibility of what that new card might be?
2. 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?
3. 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?
4. 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.)
5. 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.
6. 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 are still unanswered, and even still poorly-defined. At best they only hint at a future direction. However, I think I have begun to make some progress in answering some of them in my thesis work.

One preliminary conclusion is that transplanting rational techniques from complete 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 my ongoing research, I plan to soon 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. Hopefully I will also be able to generalize rational abstraction methods to be of some use in other large decision problems as well.

5.3 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 time, we might learn general machine-learning techniques, which we could apply to other areas.

For example, in Appendix A, 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 [10], 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.

5.4 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 [8], 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:

1. 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.
2. 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.
3. 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.

6 Conclusion

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.

7 References

[these need a little work still]

[1] Eric B. Baum and Warren D. Smith, "Best Play for Imperfect Players and Game Tree Search," unpublished draft report.

[2] Russell & Wefald, "Principles of Metareasoning"

[3] Judea Pearl, *Heuristics*

[4] Ron's thesis

[5] McAllester's conspiracy numbers paper

[6] A Hansson & Mayer paper on games

[7] Berliner reference from Pearl

[8] Barney Pell, Ph.D. thesis

[9] Warren D. Smith, "Approximation of staircases by staircases"

[10] Matthew Ginsberg, "Is there any need for domain-dependent control knowledge?"

8 Appendix A—Automated evaluation function

This appendix 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 hand-made 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. The original references for this technique have been lost; I am still tracking them down, but apparently something like this has been studied in some branch of statistics.

8.1 The Problem

We're 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 1, "Classification Spaces," on page 27). We generally try to choose a classification space that we think will provide a lot of information about the variable we're 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 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. (A la Baum and Smith.)

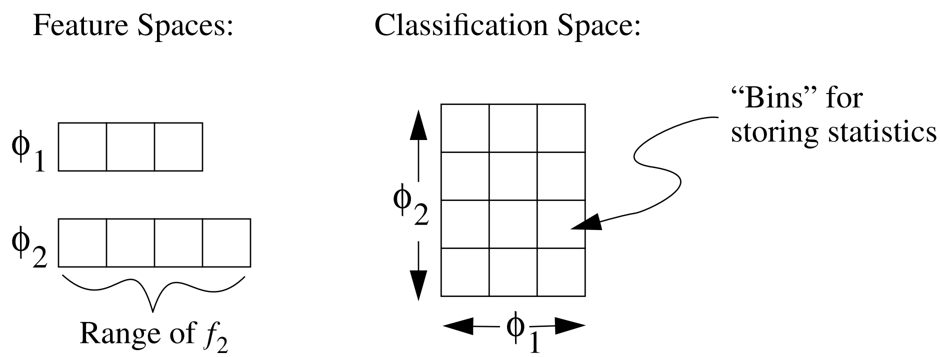
Then, for each bin, we form a model of the data in that bin, in the form of a probability distribution over y (or, in the case where additional information is stored in the bin, a function from $x_1, \ldots, x_k$ to distributions over y). We might get the distribution by choosing the maximum-likelihood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

So then we're 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.

FIGURE 1. Classification Spaces



The point of this paper is to briefly describe one possible solution to this problem. This method is not original with me, but it's been a long time since I worked on this and I don't remember the references; I'm in the process of tracking them down.

8.2 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 recommendations 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?

Well, here is one possibility. Each classification space, with probability distributions in all of its bins, can be thought of as a probabilistic model of the data, i.e., as a hypothesis that a position's outcome is determined by selecting a value from a 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.

8.2.1 Application of the T echnique

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.

8.2.2 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]

If the variable y has a small, discrete range (e.g., $\{\mathrm{Win}, \mathrm{Lose}, \mathrm{Draw}\}$), and we accept a normal flat prior over the space of possible distributions over y, then there is no problem; a standard formula[2] for the exact distribution is given by:

$$\wp(y_i) = \frac{m_i + 1}{M + n_y} \qquad \text{(EQ 11)}$$

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

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 $e^{-\#\mathrm{bins}}$. This needs more work.

2. EQ 11 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.

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 EQ 11 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 EQ 11 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.

I have thought of a few more complex ways to try to deal with this issue, but I'll save them until they're better developed.

8.3 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 classification 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.