How a Bayesian Approaches Games Like Chess

Eric B. Baum[1]
NEC Research Institute, 4 Independence Way, Princeton NJ 08540
eric@research.NJ.NEC.COM

Abstract

The point of game tree search is to insulate oneself from errors in the evaluation function. The standard approach is to grow a full width tree as deep as time allows, and then value the tree as if the leaf evaluations were exact. This has been effective in many games because of the computational efficiency of the alpha-beta algorithm. A Bayesian would suggest instead to train a model of one's uncertainty. This model adds extra information in addition to the standard evaluation function. Within such a formal model, there is an optimal tree growth procedure and an optimal method of valueing the tree. We describe how to optimally value the tree, and how to approximate on line the optimal tree to search. Our tree growth procedure provably approximates the contribution of each leaf to the utility in the limit where we grow a large tree, taking explicit account of the interactions between expanding different leaves. That is to say, our procedure estimates the **relevance** of each leaf and iteratively expands the most relevant leaves. Our algorithms run (under reasonable assumptions) in linear time and hence except for a small constant factor, are as efficient as alpha-beta.

Introduction

[Shannon, 1950] proposed that computers should play games like chess by growing a full width game tree as deeply as time permits, heuristically assigning a numerical evaluation to each leaf, propagating these numbers up the tree by minimax, and choosing as the "best move" the child of the root with the largest number. Now the whole point of search (as opposed to just picking whichever child looks best to an evaluation function) is to insulate oneself from errors in the evaluation function. When one searches below a node, one gains more information and one's opinion of the value of that node may change. Such "opinion changes" are inherently probabilistic. They occur because one's information or computational abilities are unable to distinguish different states, e.g. a node with a given set of features might have different values. In this paper we adopt a probabilistic model of opinion changes, describe how optimally to value the tree in this model, and give a linear time algorithm for growing approximately the most utilitarian tree.

Tree Valuation

We first briefly discuss how to value a given partial tree with heuristic evaluations at the leaves and then take up the more interesting question of which search tree to grow. Note that minimax, while producing best play in games between "perfect" players who can afford to search the entire game tree, is not evidently the best way to utilize *inexact* leaf values in a given partial tree. Nor is another old idea [Pearl,1984] that we call "naive probability update." Instead, from the point of view of a Bayesian, one should model one's uncertainty, and within the context of such a probabalistic model derive the optimal strategy. This leads to a propagation rule we call "best play for imperfect players," or BPIP. BPIP has a simple recursive definition.

We adopt an evaluation function which, rather than just returning a single number estimating the expected value of further play in the game, also returns a probability distribution $P_L(x)$ giving the likelihood of opinion changes in that number if the node were searched deeper. $P_L(x)$ is the probability that if we expanded leaf L to some depth, the backed up value of leaf L would then be found to be x. The mean of our distribution valued evaluation function is an ordinary evaluation function, but our distribution gives the probability of various deviations.

Such an evaluation function may be readily learned. In essence one may empirically measure the likelihood of various opinion changes as a function of various features. So for example one may adopt a set of discrete valued features. These divide[2] the space of game configurations up into a set of discrete bins. Now we take a standard evaluation function and play a large number of games using alpha-beta at, say, depth 6. For

[1] This is a super-abbreviated discussion of [Baum and Smith, 1993] written by EBB for this conference.

[2] In practice we have been dividing game space up using a feature based decision tree.

each position we encounter, we determine its bin index, and place in the bin the difference between the direct evaluation of the position and its backed up value. In the end the bins contain histograms of the deviations coming from deep search. We replace these histograms with smoothed versions. Our distribution valued evaluation of a new position is then the distribution retrieved from its bin shifted by the position's naive evaluation. The distribution in the bin is a directly measured estimate of the probability, conditioned on the feature values of the position, that any particular reestimation of the value would be found if we searched the position deeper.

We assume these distributions are independent.[3] BPIP then yields a certain formula [Palay, 1985] for propagating these distributions up the tree which associates to each node n in the tree the probability node n's negamax value is x given that a value is assigned to each leaf from its distribution.

After we are done expanding the tree, the best move is the child of the root whose distribution has lowest mean. (Lowest means best since we are negamaxing.) Note that we take means at the child of the root *after* propagating, whereas the normal (Shannon) approach takes the mean at the leaves before propagating, which throws away information. In figure 1, the Shannon approach is indifferent to which move to make, whereas BPIP makes a more informed choice.

One can calculate all distributions at all nodes exactly, while expending a computational effort, and consuming an amount of memory, depending (under reasonable assumptions) only linearly on the number of leaves in the searched tree. So, despite the fact that we use the statistical information at leaves "correctly," and use more of it, and do all our calculations exactly, we use only a small constant factor more computer time than standard methods. We do consume linear memory, whereas the standard approach consumes sublinear memory.

We[4] have experimented on the game of Kalah, comparing our distribution propagation scheme on a full width tree to minimax which propagates the means of our leaf distributions[5] on the same full width tree. BPIP wins at all depths from 0 to 9, although sometimes by small margins.

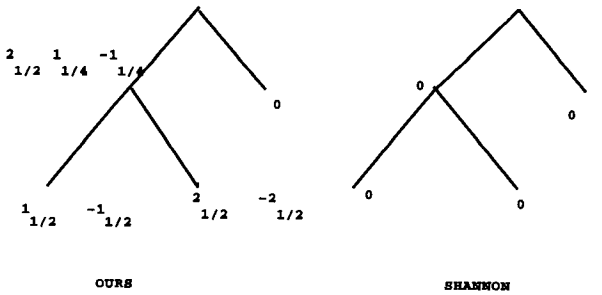

Figure 1: BPIP valuation vs Shannon valuation. The notation $(1_{1/2}, -1_{1/2})$ denotes the probability distribution: probability 1/2 of being 1 and probability 1/2 of being -1. Shannon is indifferent to which move to make. BPIP would find that the mean of the left branch's distribution is 1, and so would choose the right branch (with higher negamax value). If the left branch is chosen, the opponent would then look deeper in his turn, and would likely find that one of the two leaves is better for him than 0.

Tree Growth

The main motivation for calculating all these exact distributions is, however, to allow "utility guided tree growth." Full width exploration, that is, exploring every node in the game tree down to some cutoff depth, seems suboptimal. Surely there must be some better idea in which the lines of play which are more likely to have more impact on one's choice of "best move," are explored more deeply than the lines of play that are unlikely to have much impact. Intuitively, the likely impact of expanding a particular leaf on one's chances of winning may be said to be the "**relevance**" of that leaf. Intuitively, the relevance of a given leaf will depend, for example, on the intrinsic uncertainty in one's evaluation of that leaf (reflected in its distribution) and on the interaction with other leaves' distributions. For example the probability the leaf is on the principal variation (averaged over the probability distributions at all leaves) is intuitively a factor. By modelling our uncertainty in the leaf evaluations we are able to make intuitive notions of leaf relevance precise.

We model the expansion of a leaf as selection of one value from its distribution. (By "expansion of a leaf" we mean agglomerating its children onto the tree.) Since our distributions are designed expressly to estimate the likelihood of various outcomes when we expand the leaf, this is the natural Bayesian model[6]. This gives a formal model, and within this model, one may describe the optimal decision-theoretic strategy for growing a search tree. Unfortunately this strategy

[3] Note: we are *not* assuming that the probabilities of winning at different nodes are independent, as have [Pearl, 1984; Chi & Nau, 1989]. We are assuming that the *errors* in our estimate of these probabilities are independent.

This independence assumption is only approximately valid, in practice. One must be careful in training one's evaluation function, e.g. in choice of features, to achieve adequate independence.

[4] Experiments have been done by Charles Garrett and Rico Tudor.

[5] Alternatively, we have compared to another high power standard evaluation function with similar results.

[6] In practice, when we expand a leaf, each of its children gets an associated distribution, thus the node's distribution, instead of becoming a single spike, merely sharpens. We *model* the distribution as becoming a spike in determining its 'relevance', e.g. which leaf to expand next.

seems computationally intractable.

A practical solution is to grow the search tree by repeatedly expanding the leaves with an appropriately defined largest **relevance**. The "immediate expansion utility" of a leaf is the gain that would accrue if one expanded that one leaf and then chose one's move, rather than choosing one's move with no expansion. Expanding according to immediate expansion utility (similar to the "metagreedy" idea of [Russell & Wefald, 1991]) is not good enough because of its neglect of one's ability to expand other leaves before moving. For example, frequently a "conspiracy" among many leaves is necessary to affect the best move choice [McAllester, 1988]. Even when this is not the case, the metagreedy approximation is undesirable, see e.g. footnote below. We may similarly define the expansion utility of any subset of the leaves. In fact, we can compute in linear time the expansion utility U of expanding the entire game tree. This gives a natural condition for termination of the search: stop searching and make your move when the cost of computer time outweighs the utility of further search. This condition may be used to divide thinking time between moves, allocating more time to moves where the decision is more difficult and more important.

Another viewpoint is the following. A configuration of leaf values is an assignment of one value to each leaf of the tree. There is uncertainty about the exact value of any given leaf so there are many possible configurations. At any given time, we have some favorite move that we would make if we had to move without further expansion. For some of the possible leaf configurations, this is the correct move, and for some of them it is the wrong one. What makes it our current favorite is that it is the best on average. If we could expand all the leaves of the tree, we would make the optimal move whichever the true configuration turns out to be. Thus the expansion utility U is identically the sum, over all probabilistically weighted leaf configurations for which some move is better than our current favorite, of how much this alternative choice is superior.

This reasoning leads to the leaf **relevance** measure we call "ESS." The idea is that the **relevance** $\langle \delta_L \rangle$ of a leaf L should be the total contribution of L to *all* the 'conspiracies' it could participate in. That is $\langle \delta_L \rangle$ should be the total contribution of leaf L to U. The contribution of leaf L to a particular configuration is directly measured by asking, if you fixed all the other leaf values, how important knowledge of leaf L's value would be in predicting how much better some move is than our current favorite. So we define $\langle \delta_L \rangle$ as the probabilistically weighted sum over all leaf configurations, of the absolute value of the contribution of leaf L in each configuration. If you were going to expand all the leaves, $\langle \delta_L \rangle$ would be the expected amount of gain that you would have received from expanding leaf L. Thus $\langle \delta_L \rangle$ is the natural measure of the **relevance** of expanding leaf L if you intend to continue expanding until you are quite confident which is the best move, that is if you expect to extract most of the utility U in the tree.

Our "ESS" is a value V_L associated with each leaf that provably approximates $\langle \delta_L \rangle$ to within a factor of 2. Under some practical restrictions, described in the full paper, the V_L is identically equal $\langle \delta_L \rangle$. The ESS itself has intuitive meaning: it is the expected absolute change in U when we expand leaf L. When you expand leaf L, the remaining utility from expanding all the rest of the leaves can either rise or fall. When it falls, this takes you closer to moving, since remember the natural condition for terminating search and selecting a move is when the remaining utility falls below the time cost. When U rises, this means that the result from expanding leaf L was surprising, and that you were mistaken about the utility of expansion. Both types of information are valuable, and the ESS asigns them equal value[7]. Alternatively, the ESS can be seen as the best estimate of the *a posteriori* change in the expected utility. Thus the ESS is a natural leaf **relevance** measure in its own right, and furthermore is shown by a (hard to prove) theorem to approximate the contribution to the utility made by the leaf in the large expansion limit. We have algorithms that compute the ESS values V_L for all leaves L in our search tree, exactly, in a number of arithmetic operations depending only linearly on the size of the tree.

This, then, is our proposal: valuate the tree using BPIP, and grow it by repeatedly expanding the leaves of our current tree which have the largest ESS values. Keep re-growing and re-valuating until the utility of further growth is smaller than the estimated time-cost it would take, then output the best move. We propose various algorithmic devices to do this efficiently, e.g. the "gulp trick," certain multilinearity lemmas, and our "influence function" methods. Assume that the time to evaluate a position is comparable to $d \log b$, where b is the mean branching number and d is the geometric mean depth of the leaves in the final tree. This is entirely reasonable for complex games like Chess, where evaluation is relatively expensive, but may fail in games like Kalah, where evaluation is cheap. If this assumption holds, then our entire move-finding procedure will run in time depending only linearly on the size of the *final* search tree (that is, after all growth is complete). (If this assumption fails, then BPIP gives up a logarithmic factor).

The constant factors in our time bounds are small. Thus for chess assume that we tune our algorithm to search three times as deep along the lines judged most important as alpha-beta. Then if our algorithm and alpha-beta explore for the same amount of time, the tree we grow will be up to three times as deep, but contain about half as many leaves, as that of alpha-

[7] By contrast, the metagreedy approximation cancels these different sources of information instead of adding them.

beta.

Experiments in progress will be reported in more detail elsewhere. We have implemented our BPIP approach on Kalah. It wins about 20% more games than alpha-beta when both perform the same number of evaluations. Our alpha-beta competitor employs an extremely effective move ordering, as evidenced by the fact that my collaborators used this move ordering to mathematically solve Kalah (first player win). Our BPIP algorithm searches about a factor 3.5 fewer boards in an equal time than Alpha Beta. In the (very preliminary) experiments to date, at very low time controls (about .3 seconds per game), this constant factor appears critical, i.e. Alpha Beta wins substantially more games at equal time control. I will report on results at longer time control at the talk. We also have experimental evidence that further improvements (underway) in our evaluation function, making the distributions less corellated, might improve performance.

In summary the experimental results to date are interesting but not overwhelming. We are also implementing BPIP on Othello, and I hope to report on Othello results at the meeting.

Previous work

A number of tree growth algorithms have previously been proposed, e.g. by [McAllester, 1988; Palay, 1985; Rivest, 1988; and Russell and Wefald, 1991].

Palay was the first author to propose the use of distribution valued evaluation functions and also proposed the same equations for propagation of probability distributions that we do, but his motivation and application were different.

Russell and Wefald's contributions included enunciating optimal search as a goal, and the use of a utility based stopping condition and of evaluation functions which return distributions. Their "metagreedy" approximation of leaf expansion neglected interaction effects, and their algorithm required time superlinear in the number of leaves.

McAllester first enunciated the notion of conspiracy number and gave an interesting algorithm for constructing high conspiracy number trees. His approach basically considered the relevance of a leaf to be the smallest conspiracy it could participate in.

Rivest gave an ingenious algorithm which roughly speaking considered the relevance of a leaf to be the partial derivative of the root value with respect to that leaf, when the minimax nodes were replaced by a differentiable approximation. This approach is fascinating, but somewhat ad hoc.

From our point of view any tree growth procedure which treats the leaves of a game tree as independent, as does alpha-beta, all the above authors, and almost all computer science work on game tree search[8] is either implicitly or explicitly striving to approximate the decision theoretic optimal leaf to expand next. Our approach of stating a Bayesian model of search, and then giving a provably efficient algorithm approximating Best Play for Imperfect Players can thus be seen as unifying and formalizing this line of research. Previous heuristic ideas like "singular extensions" [Anantharaman, Campbell, & Hsu, 1990; Anantharaman, 1990], and "quiescence search" [Beal, 1990] as well as alpha-beta style cutoffs occur automatically[9] in our procedure. We review previous work in more detail in the full version.

[8] Human players seem not to make this assumption.

Pointer

We have provided this brief introduction as we are unable to coherently describe the details within the present page limitations. Details may be found in [Baum & Smith, 1993] which has been submitted for publication, and in the interim may be obtained by anonymous ftp from external.nj.nec.com, in file pub/eric/papers/game.ps.Z.

Acknowledgement

All work reported here was done in collaboration with W. D. Smith. All programming was done by Charles Garrett, Rico Tudor, and W. D. Smith.

References

T. Anantharaman, M. Campbell, F. Hsu: Singular extensions; adding selectivity to brute force searching, Artificial Intelligence 43 (1990) 99-109

Thomas S. Anantharaman: A Statistical Study of Selective Min-Max Search in Computer Chess, (PhD thesis, Carnegie Mellon University, Computer Science Dept.) May 1990, CMU-CS-90-173

D.F. Beal: A generalized quiescence search algorithm, Artificial Intelligence 43 (1990) 85-98

E.B. Baum, W.D. Smith: Best Play for Imperfect Players and Game Tree Search, submitted for publication 1993

Chi, P-C, D. S. Nau: Comparison of the Minimax and Product Back-up Rules in a Variety of Games, in *Search in Artificial Intelligence*, eds. L. Kanal and V. Kumar, Springer Verlag, New York,(1989) pp451-471.

D.A. McAllester: Conspiracy numbers for min max search, Artificial Intelligence 35 (1988) 287-310

A.J. Palay: Searching with probabilities, Pitman 1985

J. Pearl: Heuristics, Addison-Wesley 1984

R.L. Rivest: Game tree searching by min max approximation, Artificial Intelligence 34 (1988) 77-96

S. Russell and E. Wefald: Do the Right Thing, MIT Press 1991 (see especially chapter 4)

C.E. Shannon: Programming a computer for playing chess, Philos. Magazine 41,7 (1950) 256-275

[9] Similar effects occur in some previous tree growth procedures, e.g. Rivest's or Russell and Wefald's.

Commentary on Baum's "How a Bayesian ..."

Stuart Russell, Computer Science Division,
University of California, Berkeley, CA 94720.

Summary of the Paper

The paper divides the problem of game playing into two parts: growing a search tree and evaluating the possible moves on that basis. The evaluation process is based in part on the idea that leaf node evaluations should be probability distributions rather than point values, and should be propagated back to the root *before* taking expectations. The propagation algorithm is more sophisticated than the product formula derived by assuming independence of leaf values. It is, however, weaker than the method of Hansson and Mayer (1989), who allow for dependencies by formulating the search tree as a belief network. Hansson and Mayer use the probability distribution of the *true cost* of the leaf. Baum and Smith use the distribution of the backed up value after expansion (Russell & Wefald, 1988). Although this short paper does not describe the analysis, the BPIP algorithm uses this because it describes the information the algorithm expects the player to have when (or if) that node becomes the current state. The distribution is obtained empirically, thereby providing metalevel information on the results of computations. As stated, it assumes the expansion yields a point value, whereas in the Baum–Smith scheme (unlike the Russell–Wefald scheme) it actually yields a distribution. Presumably this can be fixed by a rigorous analysis.

The relevance of the paper to the symposium lies in its use of decision-theoretic metareasoning to select nodes to expand and to terminate search. A computation is relevant if it promises to provide utility in the form of improved decisions (either now or eventually). Although this idea goes back at least to I. J. Good (1968), it has proved difficult both in theory and in practice. Ultimately, the approach promises to eliminate the notion of algorithm in favour of the notion of adaptive, rational metalevel control of object-level computations.

Baum and Smith have made significant contributions to the understanding of how to find efficient approximations to rational metacontrol (which is itself intractable). They attempt to avoid myopic estimation, which was perhaps the principal problem with Russell and Wefald's MGSS* algorithm, and to improve the bookkeeping in order to cut search overhead per node. Their approximation is based on the idea that there is some total value to be obtained from *complete* expansion of the tree (to termination?). Each node expansion makes some contribution to this. They make a plausible claim to the effect that this is related to the expected absolute change in the completion utility when the node is expanded. However, the connection to the actual expansion utility of the node is unclear. The correct expansion utility can only be analysed by treating the metalevel problem as a stochastic sequential decision problem, but I could find no such analysis. Discussion of the *qualitative* properties of the expansion measure should prove interesting, and will certainly contribute to our understanding of the relevance of computations.

Relevance and the Value of Computation

The value of computation (VOC), which is strongly connected to information value theory, has some nice properties. For example, provided rational use is made of the results of computation, computations have nonnegative expected value. It can be quite confusing, however. I. J. Good, for example, suggested that a computation is valuable if it increases the value estimate of the root (see also Rivest). This rules out computations that might reveal one's plan to be a blunder—OK for politicians, but not for game-playing programs.

Part of the difficulty lies in the formulation. $P(A|B)$ should be independent of the *form* of B—i.e., any logically equivalent expression should be treated the same way—yet computations do nothing but yield logically equivalent forms. No new information is generated. Hence one has to resort to arbitrary restrictions, such as that the metalevel may not do node expansions in order to find the value of a node expansion. While one can "partition" the agent, so that the metalevel has no access to the rules of chess, this makes it impossible to show rationality of the agent as a whole.

Ultimately, there is no solution within the approach to control that tries to calculate what calculations are rational. The theory of bounded optimality offers a way out, by insisting not on rational control (which would involve arbitrary anounts of metalevel computation) but simply on whatever control results in the best overall program.

One way to achieve this is through adaptation within some architectural parameters. The value of computation can be *learned* by doing computations and seeing whether they helped. Computation sequences can be handled by formulating a reinforcement learning problem at the metalevel. One benefit of the kind of analysis in Russell–Wefald and Baum–Smith is that it helps to identify the relevant (in the inductive sense) features of the computational state so that learning can proceed quickly.

This leads to another interesting question. Consider the problem of learning the value of (say) allocating 100 nodes of search to a given type of problem. The value depends on how those 100 nodes are used, which depends in turn on the metalevel control algorithm's method for estimating the value of computation. We therefore have a feedback loop in the learning process. As yet, little analysis has been done on the stability and convergence of such processes (Tash & Russell, 1994).

Finally, one can question whether the tree-searching approach to games can be taken much further. Restricting computations to generation and evaluation of concrete states in continuous sequences from the current state seems unnecessary, although it does make for a sufficiently uniform computational process that allows metalevel analysis to proceed. More complex architectures, including condition-action rules, goal-based planning, and so on, provide much more relevant computations for the metalevel to choose from.

References

Good, I. J. (1968) A five year plan for automatic chess. *Machine Intelligence*, **2**.

Hansson, O., and Mayer, A. (1989) Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario.

Russell, S. J., and Wefald, E. H. (1988) Multi-level decision-theoretic search. In *Proceedings of the AAAI Spring Symposium Series on Computer Game-Playing*, Stanford, CA.

Tash, J., and Russell, S. (1994) Control strategies for a stochastic planner. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA.