From: AAAI Technical Report FS-93-02. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.

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. 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.

We first argue that minimax, while producing best play in games between "perfect" players who can afford to search the entire game tree, is not 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.

To implement BPIP, one needs a "probabilistic model of extra information." The words "extra information" denote the idea that an imperfect game player, upon reaching a node in the game tree during later play in the game, will then be able to search deeper below that node, and thus gain access to more information than he previously had. We adopt an evaluation function which, rather than 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. We describe a statistical method for training such an evaluation function. In essence one may empirically measure the likelihood of various opinion changes as a function of various features.

We assume these distributions are independent.[2] (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 in-

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

[2]We also make a "depth free" assumption, implicitly also made by minimax, and likely to be less important than the independence assumption.

From: AAAI Technical Report FS-93-02. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.
dependent.) BPIP then yields a certain formula for propagating these distributions up the tree. One 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 highest mean. 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.

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.

Experiments on the game of Kalah compare our distribution propagation scheme on a full width tree to minimax which propagates the means of our leaf distributions[3] on the same full width tree. A simple heuristic is added to offset in part the lack of true statistical independence. Preliminary indications are that our scheme on a tree of depth d, for d equal five to ten, is a bit stronger than minimax on a tree of depth $d+1$.

The main advantage of 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.

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 of expansion. 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 seems computationally intractable.

A practical solution is to grow the search tree by repeatedly expanding the leaves with an appropriately defined largest "expansion importance." 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 4 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.

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 importance measure we call "ESS." The idea is that the importance $\langle\delta_L\rangle$ 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 importance 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

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

From: AAAI Technical Report FS-93-02. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.

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[4]. 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 importance 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. Using various algorithmic devices we propose, in particular the "gulp trick," certain multilinearity lemmas, and our "influence function" methods, this 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).

The constant factors in our time bounds are small. Thus for example, for chess, assume reasonably that the time to evaluate a position is large compared to the mean depth of search (times a small computation time), and 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-beta.

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]. From our point of view these authors were all (either implicitly or explicitly) striving to calculate the decision theoretic optimal leaf to expand next. Our approach of stating the Bayesian model of search, and then giving a provably efficient algorithm approximating Best Play for Imperfect Players can thus be seen as unifying, formalizing, and (at least from a theoretical point of view) to a certain extent resolving 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 in our procedure. In contrast to the previous approaches, which were all greedy or ad hoc, our leaf importance measure accounts for the possible outcomes of future expansion. We review previous work in more detail in the full version. We also remark there that the algorithms of [McAllester, 1988] and [Russell & Wefald, 1991] take time superlinear in the number of leaves.

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 use of them was approximate, and his motivation and application different. Our goal of optimal search, our use of a utility based stopping condition and of evaluation functions which return distributions were stimulated by Russell and Wefald, but we believe improve on theirs.

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

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.

Experiments in progress will be reported elsewhere.

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