Experiments with a Bayesian game player

Warren D. Smith Eric B. Baum
Charles Garrett
NEC Research Institute 4 Independence Way Princeton NJ 08540
Email: {wds, eric}@research.NJ.NEC.COM; garrett@cs.washington.edu

Rico Tudor
Pattern recognition systems, 1890 maple av, suite 115, Evanston IL 60201
Email: rico@math.nwu.edu

November 27, 1996

Abstract.

In [7] we proposed a Bayesian algorithm for game playing that we will call BP. BP maintains a probabilistic model of its uncertainty and uses it to grow its search tree in the most relevant directions, and to value the tree's nodes. Here we describe the first implementations of this algorithm and report on experiments comparing it with the standard (alpha-beta, or AB) approach, and the "probability product" (PP) approach, in several popular games. BP is seen to evaluate a fixed tree more accurately than either AB or PP in a large variety of games. BP is seen to beat strong alpha-beta programs in Othello and Warri even when the alpha-beta programs are given substantially greater computational resources. We elucidate which conditions favor BP and which favor AB. We have invented several important BP-gameplayer engineering tricks in the course of this research, such as the "multispike trick" and new methods of evaluation function design. More such tricks undoubtably remain to be discovered.

1 Introduction

1.1 Three ways to valuate a tree: BP, AB, and PP

In a companion paper[7] , we proposed a Bayesian procedure for making computers play games such as chess. We call this procedure "BP". The BP approach both values a given search tree differently (each node is associated with a probability distribution, rather than a number; the move with largest mean is chosen) and grows a more selective tree (leaves with large "expansion importance" based on certain Bayesian principles, are expanded) than standard approaches.

Shannon[49] proposed that computers select their move according to the minimax value of a full width subtree, with numerical leaf values assigned by some readily computed, heuristic evaluation function. The alpha-beta (AB) procedure speeds up the minimaxing. Various heuristic improvements such as "move ordering" and "iterative deepening" allow alpha-beta to achieve in practice nearly its theoretical limit speed up, i.e. to search in a given time nearly twice as deep a full width tree as would be searched by a naive tree searcher. Other heuristics attempt to extend search along hopefully important lines or retract search on hopefully unimportant lines. Alpha-beta, with heuristic improvements, has been the search engine in virtually every high-performance game program since its invention, including games of chess, checkers, and Othello where computers are comparable to or better than human experts[1].

We also compared both alpha-beta and BP to a previously proposed probablistic scheme that we call Probability Product or PP. By PP we mean the proposal[35] to use an evaluation function estimating probability of winning, and to compute the value of a node as the probability it is a win, given the (assumed independent) estimates for its children.

1.2 Underlying assumptions and approximations

One of the approximations and assumptions that underlies BP is that the probability distributions representing evaluation function errors (or "opinion changes") at leaves are independent. This paper reports on experimental tests of that in a wide variety of games, and in some cases we find substantial deviations. Despite this, we find that BP is able to best strong alpha-beta programs at Othello and warri.

Because PP assumes the estimates themselves are independent, whereas BP merely assumes our errors in the estimates are independent, PP presumably has far more serious problems with probabilistic correlations. We report that such problems do indeed arise in our experiments. They are serious enough to likely render PP useless for games with positional features that persist for a long time.

1.3 Valuation of a fixed tree; tree growth with node count and compute time constraints

Our experiments show that BP provides more accurate node valuations than AB or PP. Specifically, if BP, AB, and PP are handed identical (full width to some depth) trees with heuristic leaf evaluations, then BP will make better use of the statistical information in those leaves to select its move. This happened in all the games we tried, at every[2] depth of tree we tried. In some games, BP's advantage over minimax is comparable to that gained by an additional ply of search.

We played games against AB where each algorithm evaluated the same number of nodes per game, but the shapes of the trees were now not identical since BP was using utility-guided growth to shape its tree. Specifically we had proposed in [7] a procedure for computing an "expansion importance" measure called "Q step size" for each leaf, and proposed iteratively expanding the fraction f most important leaves, where f is the "gulp size." Meanwhile, AB was using full-width (but $\alpha\beta$-pruned, according to a good move ordering) trees. In some cases we tried enhancing AB with "quiescence" and "probcut" [12]. BP beat AB at every game tried by this measure.

[1] In games such as Go, which computers play poorly by human standards, alpha-beta performs poorly and a new idea is apparently required.

[2] Except Othello at depth 2, where the results weren't statistically significant; see footnote 15.

Our procedure imposes a computational overhead. For games with complex evaluation functions, so that most of the time is spent evaluating leaves, this computational overhead is a small constant factor, but for simpler games, we pay a logarithmic factor[3].

We found that BP beat AB decisively in Othello and in warri – in the case of Othello, even when the competing alpha-beta program was given sizable time odds. But, AB was superior in the game of Slagle Kalah.

We played matches to determine what time odds our AB Othello program would have to be given in order to stay even with our BP program. These time odds were not only sizable, but increased rapidly with increasing time. When our BP program was given 300 seconds, the giveaway factor was already near 100. We were not able to continue by giving our BP program a time control of 30 minutes, as generally used in tournaments between humans, because at such giveaway factors our alpha-beta program would have been requiring at least days and tournaments large enough to yield reliable statistics would have required years. Instead we made a different series of time odds matches against a version of M. Buro's `Logistello`, the strongest Othello program in the world. These experiments failed to confirm that BP plays increasingly well against alpha-beta programs at longer time controls. If anything, BP's performance fell off slightly against Log-brute (§4.1.7) as the time control was increased for both sides. This may be due in part to the increasing effectiveness of engineering improvements that Log-brute had but BP and our own AB player didn't, such as its strong endgame solver and its use of transposition tables, or may be due to other, as yet unknown, effects, such as weaknesses in our evaluator that Log-brute was able to exploit.

Another interesting question is what shape tree BP will grow. Experimentally the tree is more focused than AB but but no means as focused as human game players'.

1.4 Engineering tricks

The original alpha-beta proposal has been strengthened by 40 years of engineering improvements. Achieving our results has required several engineering improvements.

1. A simple idea called the "multispike trick" §5.1 allowed us to maintain adequately detailed distributions at all levels of our tree without unduly increasing our computational overhead.

2. We have invented "Kolmogorov Smirnov trees" (KS-trees), a new statistical method (§6.1.1) that works well for constructing BP evaluation functions.

3. BP (§5.2) is found to display a sensitive dependence on the "gulp size." A gulp size of .04 improved our time odds advantage by a factor of 4 above those achieved with a gulp size of .1.

We believe that such engineering tricks and learning methods are in their infancy, and that our current program is thus subject to much improvement. Because BP uses the evaluation function both to value the tree, and to decide what shape tree to expand, it may greatly benefit from better training of its evaluation function. Our engineering tricks are described in §5 and our learning methods are described in §6. The reader interested in replicating or extending our results will find gritty details of our experiments in our (completely unpolished) Technical report [54].

1.5 Other findings

It seems to be important, during evaluation function learning, to learn from positions drawn from BP searches (the "correct" sample space), and using BP opinion changes (the "correct" values). Experiments (omitted) showed that training using game positions or positions a few random moves from games (wrong sample space), or using alpha-beta opinion changes or perfect play values (wrong data) resulted in significantly worse evaluation functions.

[3] In footnote 11 of [7] we showed how this logarithmic factor may be gotten rid of at the cost of making the BP valuation less "exact." We have not experimented with that idea.

1.6 Experimental philosophy

Our goal in this research has been to understand the capabilities of BP, rather than to produce the strongest possible game playing programs. A championship player would have to use techniques such as opening books, remembering opponent's tendencies, and thinking on opponent's time that have nothing to do with BP's value as a search strategy. We have not yet invested in such improvements.

In many of our experiments, our BP program competed against alpha-beta competitors of our own devising. We believe, and we hope to convince the reader, that these were fair tests. By competing against our own competition, we have in some sense a level playing surface. The two competitors had comparable evaluation functions – i.e. alpha-beta typically employed as evaluation function the (precomputed) mean of the BP function. Our alpha-beta programs of course employed good move ordering heuristics. There are a set of known tricks that our alpha-beta programs did not employ, but we believe these to be of substantially less importance in improving play. Where a selective search heuristic did seem to be particularly strong, i.e. Buro's "Probcut"[12] in the game of Othello, we did implement this (§4.1.5, §4.2.1) and tested BP against it.

We have also competed against other people's programs. Our alpha-beta programs have competed quite strongly against human masters and outside programs. Our BP competitors have played even more strongly against open competition. We have also played several matches between our BP Othello program and other top programs on equal hardware, with their opening books turned off (§4.1.6). These programs reflect intense effort devoted solely to engineering performance including much faster (by factors up to about 35) evaluation functions and move generators, transposition tables, and much better endgame solvers. In spite of the fact that we have not devoted effort to such improvements, our BP searcher appears to be almost as strong as the AB searcher in the (at the time) second best program in the world, and substantially stronger than the third.

1.7 Previous experimental work.

A number of authors have previously compared tree growth algorithms to alpha-beta. Rivest compared an algorithm of his devising to alpha-beta on the game of Connect 4, finding that he could beat it at equal nodes, but lost at equal time[39]. Schaeffer [46] implemented the conspiracy number algorithm of McAllester[32] and compared it to alpha-beta in chess, finding that it worked well in tactical middlegame situations, but was not competitive overall. Palay compared his algorithm to Belle on tactical positions in chess[34]; further investigation was by Berliner and McConnell[9]. Russell and Wefald[42] reported that their "MGSS*" algorithm (incorporating utility ideas) beat alpha-beta at Othello by heavy margins, but their implementation of alpha-beta used no move ordering heuristics[43]. Re-examination of their node count data with the aid of hindsight suggests that, if their AB implementation had used the same move ordering heuristics that ours did, then their MGSS* performance would, instead, merely have been comparable to AB. For more discussion of previous work, see [7].

Our results are the first of which we are aware where a non-minimaxing alternative approach was able to beat alpha-beta programs under realistic conditions.

2 Experimental methods, general discussion

2.1 Tournaments

Most of our experiments consisted of multigame matches between two gameplaying entities. Our tournaments were "color balanced" e.g. each player would play each gamestart position from each side. As gamestarts, we used:

- the positions at the ends of 71 named Othello openings (this list was posted on the internet Othello server by R. Gatliff),

- a list of 51 "reasonable" Warri openings provided by David Chamberlin and a set of the 190 warri positions reachable from gamestart in 3 ply,
- all 31 positions in Slagle Kalah with 36 stones, 3 per house except that we allow one house on each side to have 4 and one with 2 (same positions one each side), or all 961 positions gotten as above, except we remove the requirement for the two sides to be the same,
- and in other cases, simply every position reachable from gamestart in a certain number of ply.

2.2 Games, languages, hardware

We have studied the abilities of full game playing programs on three games, Slagle kalah [50], Othello [26], and warri [13]. Kalah was chosen as a simple game to begin on; Othello as a more complex game on which alpha-beta performs well; and then warri was chosen as a more complex relative of Kalah. For rules, see §7. We would like to see experiments on chess, but have abandoned these for the present paper as requiring too much programming work.

In addition to the above games, we have studied different evaluation methods on fixed sized trees on the games "mod-9 connect-4," and Pearl's P-game [35] [33]. Pearl's P-game was included in this list because it was crafted to be "pathological" and thus seemed likely to lead to insight.

For a description of the games except the P-game, see §7. The P-game is described in §3.1.

The guts of all our programs are written in `C` and `C++`,although we have also used the following languages in various places: UNIX(TM) shell, `sed`, and `awk` [61], `TCL` [63], `perl` [64], and `matlab` [62].

All our timed experiments were run on a SGI machine based on a 150 MHz IP19 processor (MIPS R4400 processor with R4010 floating point chip) with data and instruction cache sizes of 16 Kbytes each, and a secondary unified 1 Mbyte cache. All our runs fit inside 90 MBytes of RAM.

2.3 AB opponents written by us

We have played against two sorts of alpha-beta opponents, those written by us, and those written by others (§2.5).

Our own alpha-beta opponents use good move-ordering heuristics (based on some combination of knowledge, response killer lists, history heuristic) to shape and grow the tree, and use "iterative deepening"[52] for time control and enhancing move ordering. Occasionally other tricks were used. For example in warri we implemented quiescence (but it did not improve AB). We also implemented and report results against an alpha beta Othello program incorporating Buro's Probcut tree shaping heuristic. In general, however, we have stuck to simple versions of alpha-beta. We believe that fancy modifications of alpha-beta, e.g. negascout[37], buy little advantage in practice[4]. Although the "null-move heuristic" [18] can be very effective in chess, it would have had an extremely negative impact in checkers, warri and Othello, because "zugzwang" situations are rare in chess, common in checkers, and extremely common (in fact the norm) in warri and Othello. We will describe in §6 how we trained up evaluation functions. Generally our alpha-beta Program used as evaluation function the mean of the BP distributions. This allows a direct comparison.

AB used, but BP was denied (to keep the BP code simple) partial node expansion aided by a heuristic move ordering. Some preliminary experiments in Othello suggest that this might boost BP's strength substantially.

Our AB players did not use heuristic tree shaping methods, including "singular extensions," and various other kinds of heuristic search extensions and retractions (except, where discussed, for Probcut and Quiescence). These things are difficult to program well, and often do not buy very much improvement[5].

[4] See[48] for a comparative study of such modifications in chess; negascout caused a speedup of $< 10\%$. In trees with random leaf values [35], negascout is known to be asymptotically equivalent to plain AB.

[5] The best combination of search extensions found for the Deep Thought chess machine (after a huge amount of experimentation [6]) was estimated to be worth only 86 USCF rating points. 59 of these were due to threat extensions, 7 to singular extensions, and 5 to PV extensions. This translates to a 62:38 win ratio (where a draw is 1/2 a win), which is smaller than the advantage BP enjoyed over AB in our Othello experiments, but comparable to BP's advantage in our warri experiments.

2.4 Bugs?

As a matter of general philosophy we tried to use the simplest AB variant that we could (avoiding singular extensions, negascout, and transposition tables [for both AB and BP]) *specifically because we wanted it to be clear that our AB searcher was bug-free.* Since our AB searchers are only about 10 lines long, and have been heavily tested against outside opponents (including in same-depth games, intended to test the evaluator, and examining node count outputs to compare move ordering efficiencies) we are confident this goal was achieved.

Now on the other hand, we necessarily are considerably less confident that our BP implementation was bug free. Since this was the first-ever BP implementation, we tried to keep it simple, even at the cost of time or space efficiency, but it still added up to many pages of code. However, even if bugs remain in our BP implementation(s), the fact that they nevertheless managed to beat AB, is still a significant result.

Also, perhaps our heuristic position evaluators contained bugs. (In fact, it is not even clear how to define a "bug" in an evaluator. Arguably *any* evaluator that always returns a value without side effects, is "bug free.") But: Certainly our evaluators suffice to make our AB and BP programs play strongly against outside competition, and considering that there is every reason to believe (§4.3) that BP does better versus AB when the evaluator is better, if BP managed to beat AB with both sides using a buggy evaluator, that is still a significant result.

2.5 Outside Opponents

Our alpha-beta Othello program Obogon achieved an Internet Othello Server (IOS) rating 2039, higher than any IOS human at that time. We also report (§4.1.6) on matches against the top three ranked (at that time) IOS programs: `Logistello` by M. Buro, `Eclipse` by M. Giles and C. Springer , and `Bugs` by J-C. Weill.

The main weaknesses of our Othello programs as tournament players are:

1. Speed: Evaluation function slow ($\approx 20\times$) compared to `Logistello`. No transposition table. No thinking on opponent's time.

2. No opening book.

3. No top-quality endgame solver (the best programs [11, 57] find game theoretic value with 24 empty squares.)

We intentionally neglected the opening book and endgame solver since they don't matter much to our research – although they are important for tournament strength.

A version (call it `w1`) of our alpha-beta warri program, equipped with a transposition table, an opening book, 16-stone perfect endgame tables, and a self-learned evaluator beat warri expert and author Chamberlin 7 games to 0, and when he conceded the games, their perfect play values ranged from a 10 stone advantage to a 16 stone advantage. Chamberlin in turn is superior to a warri program (C, running on a Sun) written by Mark Masten, and Masten tells us that his program in turn is much stronger than a shareware PC program (C with assembly language). We suspect that `w1` is at least competitive with the World's top human warri players and with Allis's program `Lithidion`, which has won all the London computer olympiads it has entered, and may be the world's strongest warri entity[6]. However, we have never verified these conjectures.

The AB warri program that we used for our BP vs AB experiments is based on a simplified version of `w1` that has reduced endgame tables (only 9 stones), no opening book, no transposition tables. It also has a lower node rate (16000 as opposed to 120000 nodes/sec) because its evaluation function is more sophisticated; it is no longer merely two table lookups, but also combines 39 warri features via a decision tree. We tried to include a quiescence search but were unable to design one that strengthened the play of the program.

[6] `W1` achieves node rates over 10 times faster than Lithidion and has a significantly higher quality evaluation function. The only feature `Lithidion` has that `w1` does not, is the use of "proof number search" [3] on the opponent's thinking time in an attempt to solve certain moves.

Neither our BP nor our AB players utilized transposition tables[7] or "multilevel" or "lazy" evaluators[8]. We conjecture that both sides suffer equally from these omissions, but this is a subject for future research.

2.6 Statistics

Assume one is playing $2N$-game color-balanced matches, and players A and B each amass some number of wins (a draw counts as 1/2 a win). The difference in the number of wins is Δ. Assuming all games were statistically independent, one might conclude that A is "stronger than B with confidence worth $\Delta/\sqrt{2N}$ normal standard deviations."

However, our experiments indicate some danger that a game X vs Y will turn out to be the same (or almost the same) game as the game Y vs X, if X and Y are similar programs. If one views each of the N color balanced game *pairs* as independent events whose contributions to $\Delta/2$ have individual variances ≤ 1 one would conclude that "A is stronger than B with confidence worth at least

$$\frac{1}{2}\Delta/\sqrt{N} \tag{1}$$

normal standard deviations."

We have played it safe in this paper by using the more conservative, latter choice. These estimates undoubtedly underestimate the advantage of the better player. Many of the starting positions are unequal, making it harder for the stronger player to win a high fraction of games.

More confidence with less work can be obtained in games in which there is a *many-valued final score* associated with each game, because each game result represents more than one bit of information. For estimating confidence in tourneys of this type, let Δ be the difference in sums of the final scores of players A and B over a $2N$-game color balanced tourney, and let σ be the sample standard deviation in the final score difference *per game* over the tourney. Then assuming all $2N$ games were independent, one finds "A is stronger than B with confidence worth at least

$$\approx \frac{1}{\sigma}\Delta/\sqrt{2N} \tag{2}$$

normal standard deviations."

One could alternatively assume that all N game *pairs* were independent. The formula would then be $\approx \frac{1}{\sigma_2}\Delta/\sqrt{N}$ with σ_2, the sample standard deviation in score difference among game *pairs*. Dependence due to the presence of biased game starts tends to increase the value of $\sigma \cdot \sqrt{2}$ above σ_2 and hence the former estimate (2) is more conservative than the latter one. We have used the more conservative (2) throughout this paper.

3 Experiments, stage 1 – Comparison of PP, AB, and BP as statistical decision procedures

We first compared BP, minimax (AB), and Naive Probability Update ("PP") as statistical decision procedures. That is, we chose evaluation functions for the three methods to be as comparable as possible, and played tournaments where each algorithm looked at the same depth, full width tree. PP, BP, and AB play identically at depth 1 throughout.

[7] These allow one to avoid re-searching positions that have been searched before, and in iteratively deepened alpha-beta search can help with move ordering.

[8] Lazy evaluators have a controllable tradeoff between statistical accuracy and time consumption. These can be used to save time in alpha-beta search by calling the imprecise evaluator when it suffices to cause a cutoff. These could also be used in BP search in various ways. A "multilevel" evaluator would evaluate leaves cheaply (e.g. only counting material), but would also add a more expensive (e.g. positional correction) perturbation to the looked ahead value at 1 or 2 levels above the leaves. The result would be both fast and smart. This idea has been quite unexplored, historically.

3.1 Pearl's "P-game"

This game was designed by Pearl[35] and studied by Nau[33] as an example of a theoretically "pathological" game[9], i.e. a game where searching deeper can be shown to give smaller probability of making the correct move for some depth pairs. The game tree is full binary to some depth (11 in our experiments) and the leaves are independently randomly assigned Boolean values. Our leaves received value 1 with probability .63, picked to make the probability the first player wins with perfect play near .5.

As a heuristic evaluation function in the P-game, we used a choice suggested by Pearl and called by Nau "e2." Let $r(x) \equiv 1 - x^2$. Let node η be height h above the leaves, and let f be the fraction (assumed given to us)of these 2^h leaves with value 1. Then $e2(\eta) \equiv r^{[h]}(f)$. Here the superscript denotes functional iteration. $e2(\eta)$ is the probability that node η is a perfect-play win, given that its leaf descendants are 1 with probability f.

Recall that the BP distribution measures the likelihood of "opinion changes" as a node is expanded "further." To approximate BP, we must choose a definition of how much further to expand in producing our evaluation distributions. In the limit we expand depth 0, the BP distribution is a single spike and BP is identical to minimax. In the limit where the node is expanded to infinite depth, BP is identical to PP. We estimated the distribution assuming expansion of depth 1. Thus we used for BP a two spike distribution, one spike assuming the f given for η would also hold for its children, and the other assuming that f at the children would fluctuate by one standard deviation. Thus we took a spike of height .3 at $-r^{[h-1]}(f)$ and a spike of height .7 at $-\max_{\pm}\{r^{[h-1]}(f \pm \sqrt{2^{-h}(1-f)f}\ )\}$.

We played color balanced round robins among depth-k AB,PP, and BP players on 100,000 P-games, for[10] $k = 2, ...9$. The results were as follows.

player	depth	2	3	4	5	6	7	8	9
wins for	AB	196178	196763	184547	186982	172071	180741	172815	187157
" "	PP	199926	197412	203727	202399	211349	207292	211538	203939
" "	BP	203896	205825	211726	210619	216580	211967	215647	208904

This confirms Nau's [33] results that PP is a superior decision procedure to minimax, searching to fixed depth in the P-game. BP is found to be superior to PP with 4-9 standard deviations of confidence, depending on which depth. The advantages are small in an absolute sense.

In a later experiment, we tried playing BP with our usual Bayesian procedure for selecting the best move (expectation value) versus BP with a non-Bayesian move chooser[11], based on maximizing the probability that the selected move is in fact the one which, on deeper search, when all distributions sharpen to a single point, will be revealed to be the best move. (The probability that move 1 will prove superior to move 2 is readily computed as an integral of density$_1 \times$CDF$_2$.) $2 \times 100,000$ P-games were played at each depth, and BP won every tournament. The table gives the number of wins for BP minus the number of wins for non-Bayes at depths 2-9.

	depth=2	3	4	5	6	7	8	9
win count difference=	28874	23151	31826	28907	30277	32976	28548	28901.

3.2 Mod-9 connect 4

Mod-9 connect 4 is the Milton-Bradley game "connect 4," but now played on a (larger) 9×9 board with horizontal cylindrical wraparound. We played AB vs. PP from all possible inequivalent 3-ply starts. We did not design a BP evaluation function. Our evaluator utilizes 10 features with weights trained by linear regression. Experiments showed it estimates probability of winning well, indeed well enough to be indistinguishable from perfection by a chi-square test on a 152 game test set. Results were as follows.

[9] AB with 1, 2, or 3 extra plies bested AB in our tournaments, but AB variants that selected their move based on various weighted averages of the AB values at depths 1-d of the moves, beat plain AB at depth d (d = 4-9), although remaining inferior to depth-d PP.

[10] At $k \geq 10$ the players are perfect.

[11] This is related to an idea of Palay's, which we had reviewed in [7], §7.3

player	depth = 1	2	3	4	5	6
wins for	AB	29	44	33	45	39	52
" "	PP	29	14	25	13	19	6

For calibration, we played alpha-beta vs alpha-beta at increased depth, this time using 4-ply gamestarts.

player	depth = 1	2	3	4	5	6	7
wins for	AB	89	133	120	126	128	132	121
" "	AB+1	241	197	210	204	202	198	209

The advantage minimax has over PP is much greater that the advantage one gets by giving minimax an extra ply.

We have examined game trees to determine why PP does so poorly here. Frequently there will be a node η with some feature, such as a 3-in-a-row threat, that tends to persist. Hence many of the descendants of η have this feature, making them all slightly advantageous for one side, say evaluation .7. Say there are 1000 such descendants. PP treats these probabilities as independent (ignoring the fact that they all come from the same feature) and thus winds up computing an alleged win probability for η that may be $1-.3^{1000} = .999...9$. We expect this correlation phenomenon will devastate PP in any game with long term features.

3.3 Slagle kalah

We played PP against AB where both sides used Henry Cejtin's simple probabilistic evaluator[12] with $G = 1/2$.

	depth	1	2	3	4	5	6	7	8	9	10	11	12
wins for	AB	849	966	1105	1121	1119	1050	1024	901	915	752	26	26
" "	PP	849	853	643	690	643	725	753	869	851	983	32	33
	draws	224	103	174	111	160	147	145	152	156	187	4	3

AB has the advantage at depths 2-9, but PP wins at depths 10-12.

PP does this well because Slagle Kalah exhibits few recognizable features which last for longer than a few ply (and even these are invisible to the crude evaluation function we are using here), so that all positions are fairly "independent" of all other positions. Chi and Nau [15] showed that PP was superior to AB at certain search depths in a reduced version of Slagle Kalah. They argued that PP tends to do better against AB, if the evaluator used has a large "rate of heuristic flaw" – as do all known evaluators in Slagle kalah.

We then constructed a Slagle kalah evaluator that returned distributions. This evaluator was based on combining some kalah features via a KS decision tree (§6.1.1). The alpha-beta player used the mean of BP's evaluation function. The results are in table 1.

For comparison we played our AB player against AB with an extra ply.

	depth	2	3	4	5	6	7	8	9	10	total
wins for	AB	528	688	631	724	713	722	705	718	744	6173
" "	AB+1	1287	1134	1146	1056	1070	1041	1063	1012	986	9795
	draws	107	100	145	142	139	159	154	192	192	1330

BP's advantage in decision making quality at equal depth seems nearly as much as an extra ply of AB.

BP and PP do not use directly comparable evaluation functions. We played a tournament using different evaluation functions[13], which nevertheless seemed to have about the same strength (as judged by a negamax tournament or by play at depth 1). It is unclear how to evaluate the results of such a tournament. Totaling depths 2-7: BP won 218, PP won 156, with 23 draws.

[12] This evaluation is the exact probability of winning given the current score difference (what [50] called Kalah difference) under the assumption that you will win the seeds on your side with probability G and those on your opponent's side with probability $1 - G$, the probabilities for each seed being assumed to be independent.

[13] BP used an older evaluation function not based on K-S trees.

Depth	AB wins	BP wins	Draws	**Conf.**(σ)	AB wins	BP wins	Draws	Conf.(σ)
2	550	1286	86	11.87	871	970	81	1.60
3	736	1059	127	5.21	887	913	122	0.42
4	708	1093	121	6.21	860	935	127	1.21
5	733	1043	146	5.00	833	932	157	1.60
6	752	1045	125	4.73	885	894	143	0.15
7	766	993	163	3.66	881	883	158	0.03
8	777	991	154	3.45	862	921	138	0.95
9	808	942	172	2.16	844	905	173	0.98
10	788	953	181	2.66	340	354	87	0.35
total	6618	9405	1275	14.98	7263	7707	1186	2.47

Table 1: Slagle kalah results at equal depth. Left half: using KS-tree evaluator, multispike trick, (≥ 3) spike eval. Right half: using an older (non-KS) decision-tree based evaluator with 2 spikes always (depth 10 tourney incomplete, due to machine crash).

3.4 Othello

Depth	AB Wins	BP Wins	Draws	AB mean Discs	BP "	disc stddev
2	72	66	4	32.65	31.35	12.61
3	62	76	4	30.39	33.61	11.45
4	50	86	5	30.18	33.82	9.13

Although minimax was better than BP at depth 2, it was only by 0.61σ based on disc count (i.e. 73% confidence), and even less based on win count. The BP evaluator had been trained on opinion changes at depths 5 and 6, which are of little relevance in a depth 2 search. At depths 3 and 4, BP has the advantage, with respectively 1.68σ and 2.36σ (i.e. 95% and 99% confidence) based on disc count.

Note that both here and in the P-game (§3.1), BP's advantage over AB seems to be increasing at higher search depths; but the opposite seems to be the case in Slagle Kalah (§3.3), while the situation is unclear in warri (§3.5).

3.5 Warri

	depth=1	2	3	4	5	6	7	total 2-7
PP wins	171	72	131	119	129	195	149	966
AB wins	171	272	229	229	235	159	218	1513
draws	38	36	20	32	16	26	13	181

Here both used as their estimate of "my probability of winning,"

$$\frac{1}{2} + \frac{M - Y}{2(P+1)} \tag{3}$$

where M is the number of stones in my treasury, Y in yours, and P is the number of stones remaining in play. This quantity was truncated to lie in $[0, 1]$ and[14] if ≤ 16 stones remained in play the exact game value (from an endgame table) was used instead. Gamestarts are all 190 positions reachable in 3 ply.

Players are identical at depth 1. AB beat PP significantly at every search depth ≥ 2, except for depth 6 where the result is not statistically significant[15].

Next, we ran tournaments at fixed search depths between BP and AB, both sides using the full width tree and an evaluator based on a KS decision tree (§6.1.1) trained on positions two random moves away from positions actually found in games.

[14] This simple evaluator was found, in negamax vs. negamax testing, to be about equal to Henry Cejtin's evaluator.

[15] The reader is cautioned to remember that if you report large numbers of tournament results, as we are, it is to be expected that some few of the results will fluctuate by a standard deviation or two.

depth	AB wins	BP wins	draws	AB avg stones	BP stones	stddev stones	confid [games]	confid [stones]
2	34	34	4	24.71	23.29	6.66	0.5	-0.90
3	30	38	4	22.35	25.65	8.81	0.67	1.59
4	31	36	5	22.79	25.21	5.96	0.42	1.72
5	31	36	5	23.29	24.71	5.77	0.42	1.04
6	23	42	7	22.13	25.88	6.63	1.58	2.40
							(in normal	stddevs)
combined	149	186	25	23.05	24.95		1.38	2.6

Total stone count gives 99% confidence that BP is the superior statistical decision procedure.

3.6 Statistical dependencies potentially dangerous to BP

To examine how poorly the BP independence assumptions are satisfied in practice, we computed the "opinion changes"

$$\delta_1 \;=\; \text{Backed up BP value} \;-\; \text{Value without search} \tag{4}$$

and δ_2 (defined similarly but for a node which is a sibling of the node that yields δ_1) for ≈ 50000 pairs of sibling nodes from BP search trees. We were using KS tree (§6.1.1) evaluators.

The observed centered correlation coefficients between δ_1 and δ_2

	lookahead depth	1	2	3	4	5
Slagle kalah	centered correl. coeff.	.341	.407	.452	.443	.472
Othello	centered correl. coeff.	.410	.396	.379		
Warri	centered correl. coeff.	.202	.157	.222	.230	.259

were definitely nonzero!

We next divided the pairs of siblings into two types:

1. pairs in which both siblings fell into the same "bin" of the decision tree evaluator, and
2. pairs from different bins.

In Slagle kalah, two random positions in the search tree would have fallen into the same bin only 0.27% of the time, but siblings fell into the same bin 30.7% of the time. (Othello: 1.43% and 26.98%; Warri: 0.59% and 7.96%.) The centered correlation coefficient measured for kalah siblings of type 1 at depth 5 was .639, while for siblings of type 2, it was only .387. Meanwhile in Othello at depth 3, same bin siblings had cc= .520 while different bins were .326, and in warri at depth 5, it was same bins .459, different bins .241.

Siblings are commonly in the same bin, and when this occurs are highly correlated. Perhaps both siblings should have been rated higher than (or lower than) the usual members of their bin, for some common reason. Presumably correlations could be alleviated by simply putting more and more bins in the evaluator, e.g. by using our automated KS-tree learning method (§6.1.1) with more and more data. But there is still significant correlation even for siblings from different bins.

3.7 Conclusion and Discussion

BP's independence assumptions are significantly violated. We conjecture that this may be caused in part by an effect we call the "invisible rooks effect." Say your evaluation function does not know about some feature correlated with winning that is long term, i.e. tends to persist for several moves. For example, in chess, say the evaluation function did not know how many rooks each player had. Then the evaluation function would err on a position, and on most of its descendants in the same way, causing correlations. In a game as complicated as chess, inevitably important features will be left out of the evaluation function. This problem

will, however, diminish the better the evaluation function is. And of course, minimax and PP suffer from related problems.

Notwithstanding the violation of independence, BP won every tournament at every depth[16] against both minimax and PP, and thus appears to be the superior statistical decision procedure.

BP's advantage was sizable in absolute terms in Othello, warri, and Kalah, being almost worth an extra ply in the latter. PP was able to beat minimax at some atypical games, such as the P-game and Slagle Kalah. However in games with long term features, such as Connect 4 and warri (presumably also Othello) PP is much worse than minimax, apparently because its neglect of correlations causes PP to evaluate many nodes as near certain wins that are not.

4 Experiments, stage 2 – Comparison of BP and AB players with limited computational resources

In this section, we present experiments with more realistic gameplaying programs that utilize both the AB (minimax) and BP paradigms.

4.1 Othello

We report tournament results comparing our AB and BP Othello players at equal time limits. Since our BP player won these, we then played them at time odds in order to quantify BP's advantage.

AB and BP are using the same evaluation function (AB is using the mean of the BP function, since AB requires a scalar) and AB is using response killer history tables and iterative deepening to do move ordering (similar to `BILL` [28]). The evaluator used a combination of linear regression and KS-trees (§6.1.1). In the timed games, AB did iterative deepening until cumulative time consumption exceeded a fixed fraction of the time budgeted for that move (except that on forced moves, which are rare, it plays instantly). We are using Gatliff's list of 71 named Othello openings as our gamestarts ($2 \times 71 = 142$ games/tourney) and both players resort to a perfect endgame solver with 13 empty squares.

4.1.1 Equal time tourneys

Time	AB Wins	BP Wins	Draws	AB Discs	BP Discs	Stddev	Conf.	**Disc Conf.**
100	23	109	10	27.09	36.91	7.10	5.10	8.24
200	19	118	5	25.41	38.59	6.44	5.87	12.20
300	20	118	4	26.14	37.86	6.63	5.82	10.53
400	17	119	6	25.88	38.12	5.66	6.05	12.89

Time	AB evals/game	BP evals/game	AB time used	BP time used
100	193666 (±23099)	124082 (±20356)	84.58 (±7.13)	81.35 (± 9.18)
200	373788 (±45823)	229691 (±45143)	162.72 (±15.04)	161.10 (±25.65)
300	567280 (±64039)	337969 (±69211)	246.69 (±19.18)	244.56 (±40.03)
400	735725 (±65745)	432076 (±88376)	313.37 (±25.28)	316.92 (±53.69)

Table 2: Othello results at equal time usage.

Key:

- Time - Maximum allowed amount of thinking time (in seconds) for each player, per game.

[16] With the exception of Othello at depth 2, where minimax had a statistically not significant edge.

- AB or BP wins - The number of games won by the player.
- Draws - The number of drawn games.
- AB or BP Discs - The mean number of discs owned by the player at the end of the game.
- Stddev - The standard deviation in the number of discs owned by each player at the end of the game.
- Conf. - The number of σ of confidence that BP is stronger than AB, based on win counts and (1) of §2.6. (A negative sign means AB is stronger.)
- Disc Conf. - The number of σ of confidence that BP is stronger than AB, based on the number of discs won and (2) of §2.6. Usually more confidence is obtainable in this way.
- AB or BP evals/game - The mean and standard deviation of the number of positions evaluated by each player in a whole game.
- AB or BP time used - The mean and standard deviation of the time actually consumed by each player over the course of a game.

To summarize: In each tournament, BP won more games and more discs while consuming approximately equal thinking time. This advantage increased in tourneys in which both players were allowed more thinking time, until in 400 second games, the longest ones we ran, the win ratio was over 6 : 1. But this is still 5 times faster than the tournament time controls typically used by humans.

The fraction of time BP spends propagating distributions and deciding which leaf to expand next can be considered as computational overhead compared to an alpha-beta program. This overhead fraction decreases as the evaluator becomes slower because a higher fraction of the time is spent by both competitors on evaluations. If our evaluation function and move generator could somehow both be made 25 times faster, then our BP Othello player would still dominate AB (cf. §4.1.3), but its edge would be smaller.

4.1.2 Tree statistics

We report statistics about search tree shape – the number of leaves at each depth.

	depth																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	total
200s	3	41	367	657	1194	1496	1726	1465	1338	918	716	420	270	136	77	34	16	8	3	10885
300s	3	42	379	750	1467	1974	2411	2168	2076	1554	1271	784	552	282	175	77	42	12	4	16023

These counts are averages over all the BP search trees during a 200(300)-second tourney (rounded to integers). They may not represent any particular tree well. In the 200-second tourney, the shallowest BP tree was only 1 ply deep and the deepest was 21 ply deep. (Figures at depths 20 and 21 are not in the table because after rounding to integers we got 0. As for the 1-ply deep trees, remember that BP often chooses to get out of the search early if there is little utility in it.) Three *actual* BP tree profiles were:

	depth													
	1	2	3	4	5	6	7	8	9	10	11	12	13	
tree#1	5	12	337	378	425	1036	1554	1281	729	339	0	0	0	depth along true line=8
tree#2	2	51	470	119	58	0	0	0	0	0	0	0	0	depth along true line=3
tree#3	1	20	292	393	782	307	748	268	686	262	151	37	0	depth along true line=9

By the "true line" we mean the actual line later taken in play. The leaf-count profiles of AB's search trees are of course proportional to Kronecker delta functions (except for rare game-end leaves). The average of the AB leaf profiles during the 300 second tournament was

depth	5	6	7	8	9	10
AB#leafs(300sec)	22	1415	12580	6285	1842	265

In the 300-second tourney, the average depth of a BP leaf was 8.19, and true line was searched by BP an average of about .1 ply deeper still. In contrast, the average depth reached by the AB player was 7.45. Hopefully BP's search depth along the true game line was often deeper than AB's in situations where the move decision was difficult and important; but in other situations, it is not clear that BP necessarily will search the true game line deeply. (In fact in Warri §4.2, BP searched more shallowly than AB both on average and in true line.) Leaves expanded by BP are not explicitly chosen because it thinks that they are likely to be encountered during play. For example, if there are two moves X and Y which BP thinks are of nearly equal value while all the rest are much worse, and BP explores X for a while and decides that it stinks, then it can immediately decide to make move Y. In an earlier 100 second tourney, there were 5 games where one of the BP search trees only agreed with the true line for one move. That is, BP decided to make a move after expanding some of its siblings but not the move itself. BP won all of those games.

Note: average leaf depth figures for BP are difficult to interpret. For example, when BP decides to move in some easy situations with very little search, and save the time to search more deeply in other positions, that pulls its average depth lower[17].

4.1.3 Tourneys played at time odds

In the tables below, the columns not defined in §4.1.1 are

```
AB limit        Thinking time allotted to AB        (seconds/game)
AB used         Thinking time actually used by AB   (seconds/game)
AB ply          Average depth of AB search trees
BP used         Thinking time actually used by BP (seconds/game)
```

The first line in each table is equal time consumption.

The more time BP has to move, the greater its advantage over AB seems to be. If we assume that a factor of 4 in consumed time will give AB an extra ply of search, then we can, in a sense, translate these time figures into the amount of extra ply AB must be given to achieve parity. If BP is given 50 seconds, AB needs about 1.14 'extra ply' to achieve parity, but if BP has 100 second time limit, AB needs 2 'extra ply'. At 200 and 300 seconds we could not afford to let AB run long enough to achieve equal performance, so we must extrapolate the giveaway factors from the existing data and they could be off by as much as a factor of 2. That means that at 200 seconds, AB requires between 2 and 3 extra ply and at 300 seconds it needs between 2.8 and 3.8 extra ply.

4.1.4 Growth termination condition

We have evidence that our termination condition (§4.5) for BP tree growth, based on trading off $U_{\text{all leaves}}$ versus the cost of time, makes a positive contribution towards playing strength in Othello. Specifically, we ran a 100-second per game tournament, twice, but with the BP player's search termination logic turned *off* in the second run and replaced by logic that simply terminated search "gulping" after fixed amount of time per move had been consumed. (The AB player's time control was adjusted too, to equalize time used.)

As table 4 shows, this lowered the performance of the BP player by about 1 disc per game, although it was still much stronger than the AB player[18].

4.1.5 AB with "probcut"

Michael Buro[11] invented a simple but (at least in Othello) effective heuristic for selective extension in AB. Buro's Othello program `Logistello`, at 30 minutes/side/game would normally search to about 11-12 ply doing a nonselective iterated deepening alpha-beta search. With probcut, at each node 8 ply above the leaves

[17] Also recall that alpha-beta is using partial node expansion. BP augmented with partial node expansion would presumably yield greater average depth figures than presently.

[18] Even more statistical confidence – about 4σ – arose in earlier experiments with a weaker BP player that was more closely matched with its AB opponent.

BP with 50 second time limit (giveaway factor ≈ 5)						
AB limit	AB used	AB ply	BP used	Ratio	Conf.	**Disc Conf.**
70	47.25	5.87	36.07	1.31	4.15	6.71
150	106.89	6.46	36.96	2.89	1.25	0.26
250	181.34	6.88	36.45	4.98	0.95	0.29
450	317.10	7.27	36.53	8.68	-0.89	-2.45
BP with 100 second time limit (giveaway factor ≈ 14)						
AB limit	AB used	AB ply	BP used	Ratio	Conf.	**Disc Conf.**
120	84.58	6.26	81.35	1.04	5.10	8.24
200	144.24	6.69	82.42	1.75	3.98	6.25
400	282.99	7.22	83.11	3.41	2.61	4.43
800	552.43	7.67	82.43	6.70	1.48	1.95
1100	755.92	7.90	83.11	9.10	1.35	1.53
1600	1102.66	8.16	83.89	13.14	0.59	0.69
2100	1438.70	8.36	84.10	17.11	0.24	-1.31

BP with 200 second time limit (giveaway factor ≈ 28†)						
AB limit	AB used	AB ply	BP used	Ratio	Conf.	**Disc Conf.**
225	162.72	6.81	161.10	1.01	5.87	12.20
400	289.51	7.22	166.40	1.74	5.36	9.47
800	557.05	7.69	164.70	3.38	4.33	7.22
1600	1100.24	8.14	167.79	6.56	2.79	4.76
3200	2190.19	8.63	169.06	13.0	1.60	2.67
4800	3205.77	8.86	167.58	19.1	1.25	1.86

BP with 300 second time limit (giveaway factor ≈ 95†)						
AB limit	AB used	AB ply	BP used	Ratio	Conf.	**Disc Conf.**
340	246.69	7.10	244.56	1.01	5.82	10.53
600	418.35	7.49	239.67	1.75	5.16	9.73
1500	1032.08	8.09	245.04	4.21	4.27	8.11
3000	2033.11	8.57	245.34	8.29	3.09	5.16
6000	3943.10	8.98	247.24	15.95	3.20	4.72

Table 3: Othello results at various time odds. (See also figure 1.) †These giveaway values are extrapolations which could easily be off by a factor of 2 either way.

S.T. logic	Time	AB Wins	BP Wins	Draws	AB Discs	BP Discs	Disc Stddev	Conf.	**Disc Conf.**
on	100	23	109	10	27.09	36.91	7.10	5.10	8.24
off	100	37	100	5	28.23	35.77	7.25	3.74	6.19

S.T. logic	AB evals/game	BP evals/game	AB time used	BP time used
on	193666 (±23099)	124082 (±20356)	84.58 (±7.13)	81.35 (±9.18)
off	161397 (±17079)	112801 (±7675)	70.89 (±10.03)	71.27 (±3.81)

Table 4: Othello results – BP search termination logic turned on & off. The first line is the same tourney as line 1 of table 2.

it does a 4 ply AB search to get a value v_4. It then estimates the result v_8 of an 8 ply search as a fitted linear function $v_8 \approx av_4 + b$ of the 4 ply result and arbitrarily cuts off all nodes whose v_8 estimates lie $\geq X$ standard deviations[19] outside the alpha-beta window. It then re-searches the moves that were not cut off, to the full depth 8. The depths "8", and "4" and the optimum value $X = 1.50$ were found empirically. Probcut allows `Logistello` to search deeper in the selected lines. Its winning percentage against the nonselective version was 64.7% even in a tournament with 2:1 time odds. Also, Buro found [12] that 12 ply searches with selectivity turned on would make the same move, 93% of the time, as full width 12-ply searches, but run 4-6 times faster. Probcut is easy to implement.

We implemented a probcut version of our AB Othello player. Because our Othello tourneys were played at faster time controls than `Logistello`'s (between 50-300 seconds/side/game, as opposed to 1800) and also since our program's node rate is 25× slower, we were unable to use Buro's preferred values $(8, 4)$ for the two special heights, and instead used $(4, 2)$. First, we computed a table of standard deviation estimates indexed by number of empty squares. Then we searched for good values of X in many 100 second tournaments and found that there appeared to be 2 locally optimal values, $X \approx 0.7$ and $X \approx 1.4$. At longer time limits, we used a narrow range of X values around these two optima. As is evident from the tables 5 and 6 below, at 100 and 200 seconds AB+probcut with various finite values of X was always better than AB alone ($X = \infty$). However, at 300 seconds (table 7), small values of X actually hurt AB and only the larger values around 1.4 gave noticeably better performance than plain AB. Overall, AB+probcut remained significantly worse than BP for any value of X that we tried.

X	ABP wins	BP wins	draws	ABP avg discs	conf.	**disc conf.**	ABP avg leaf depth
0.6	28	104	10	27.42	4.51	8.54	7.04
0.7	**36**	**99**	**7**	**28.32**	**3.74**	**5.90**	**6.98**
0.8	31	102	9	27.00	4.21	8.45	6.97
1.3	29	110	3	26.70	4.81	8.71	6.81
1.4	32	107	3	27.60	4.45	7.44	6.80
1.5	30	107	5	27.13	4.57	8.26	6.79
∞	23	109	10	27.09	5.10	8.24	6.26

Table 5: Othello results at 100 sec/side/game; AB+probcut vs BP. "Conf." is the number of σ worth of confidence that BP is stronger than ABP(X); "disc conf." is the same thing, but based on disc count instead of win count. $X = \infty$ corresponds to plain AB without probcut.

4.1.6 Results against independently written adversaries

We played matches between our BP and AB programs and four independently written Othello programs. In each of these tournaments, our adversaries had their opening books turned off, and were set not to think

[19] Standard deviations in $v_8 - (av_4 + b)$, that is.

X	ABP wins	BP wins	draws	ABP avg discs	conf.	**disc conf.**	ABP avg leaf depth
0.6	25	113	4	26.50	5.22	10.72	7.58
0.7	28	109	5	27.30	4.81	8.07	7.56
0.8	24	114	4	26.91	5.34	8.77	7.53
1.3	**29**	**107**	**6**	**27.85**	**4.63**	**7.36**	**7.35**
1.4	27	111	4	26.73	4.98	9.50	7.34
1.5	29	106	7	27.63	4.57	7.82	7.33
∞	19	118	5	25.41	5.87	12.20	6.81

Table 6: Othello results at 200 sec/side/game; BP is superior to AB+probcut.

X	ABP wins	BP wins	draws	ABP avg discs	conf.	**disc conf.**	ABP avg leaf depth
0.6	19	117	6	26.61	5.82	10.38	7.89
0.7	18	119	5	26.44	5.99	10.90	7.87
0.8	19	118	5	27.25	5.87	10.33	7.83
1.3	**26**	**106**	**10**	**28.11**	**4.75**	**6.84**	**7.70**
1.4	23	114	5	27.61	5.40	8.30	7.71
1.5	28	107	7	27.51	4.69	7.67	7.67
∞	20	118	6	26.14	5.82	10.53	7.10

Table 7: Othello results at 300 sec/side/game; small X values can hurt Probcut's performance, but larger ones still help.

on our time. Otherwise they played at full strength. These four adversaries were a program of David Slate, `Bugs` by J. C. Weill, with IOS rating 2391, `Eclipse` by M. Giles and C. Springer, with IOS rating 2614, and `Logistello` by M. Buro, with IOS rating 2771. `Bugs`, `Eclipse`, and `Logistello` were the top three programs on the IOS. Slate's program is based on a comparatively fancy, full width alpha-beta type search with transposition table and quiescence, and it has a node rate $\approx 5\times$ higher than our programs. On the other hand, its evaluation function is comparatively simple. Slate's program had lost an earlier match with `BILL` [28] by a small margin.

program	wins	mean discs	sec/game consumed
AB	94	36.35	217.22 ($\pm$19)
Slate	44	27.65	227.78 ($\pm$18)
	(4 draws)	(stddev 7.58)	(disc conf. 6.83)

program	wins	mean discs	sec/game consumed
BP	131	41.88	226.99 ($\pm$23)
Slate	10	22.12	239.26 ($\pm$41)
	(1 draw)	(stddev 6.51)	(disc conf. 18.09)

Table 8: Results versus Slate's Othello program.

`Bugs` has a strong hand tuned evaluation function, fancy alpha-beta search with transposition table and quiescence, and a special purpose endgame solver. On our machine, it runs at 25000 nodes/sec – over $10\times$ faster than our programs.

Table 9 shows that `Bugs` is stronger than our AB program, but weaker than our BP program, at 300 sec/side/game. Bugs seems to have a better time control algorithm than our programs, since it uses up more of its allotted thinking time. If we give `Bugs` less time so that it actually consumes roughly the same amount of time as our AB program, AB gets better results, but still loses the tournament[20]. BP with 5

[20] Although by a statistically insignificant margin.

program	wins	mean discs	sec/game consumed
AB	49	30.96	230.62 (±18)
Bugs	89	33.04	293.53 (±4.5)
	(4 draws)	(stddev 7.16)	(disc conf. −1.72)

program	wins	mean discs	sec/game consumed
BP	106	36.65	246.23(±38)
Bugs	27	27.35	292.97(±5.4)
	(9 draws)	(stddev 5.86)	(disc conf. 9.46)

Table 9: Results versus Weill's Othello program "Bugs."

minutes of thinking time per game remains stronger than `Bugs` even if `Bugs` is given 20 or 30 minutes. This is documented in table 10.

program	wins	mean discs	sec consumed
AB	60	31.65	230.52 (±16.38)
Bugs	73	32.35	232.86 (±6.75)
	(9 draws)	(stddev 7.90)	(disc conf. −0.52)

program	wins	mean discs	sec consumed
BP(5 min)	83	33.87	250.55
Bugs(20 min)	47	30.13	1133.81
	(12 draws)	(stddev 6.18)	(disc conf. 3.60)

program	wins	mean discs	sec consumed
BP(5 min)	79	33.34	251.73
Bugs(30 min)	56	30.66	1677.77
	(draws 7)	(stddev 6.49)	(disc conf. 2.46)

Table 10: Time odds results versus Weill's Othello program "Bugs."

Eclipse features a full-width PVS search with a 2^{19}-entry trans table, and bitboard move generation. Its incrementally computed evaluator is based on tables of 5-10 square patterns. The table entries are precomputed values of a function learned by a 150-dimensional regression. `Eclipse` achieves speeds (on our 150 MHz SGI machine) of 28500 evals/sec. Finally, `Eclipse`'s special purpose endgame Win/Loss/Draw solver solves 21-23 empty squares in a few minutes. Eclipse's results vs. BP are in table 11.

Time	Eclipse	BP	Draws	B/E sec consumed	Eclipse Discs	Conf.	**Disc Conf.**
100	72	63	7	?	32.11	−0.53	−0.18
200	71	62	9	.95	32.17	−0.53	−0.32
300	71	57	14	.83	32.74	−0.59	−1.35
900	65	58	16	.84	32.44	−0.42	−0.96
1200	67	57	18	.73	32.62	−0.59	−1.24

Table 11: BP vs. Eclipse. (3 games missing at 900 sec due to machine crash.)

BP is weaker than `Eclipse`, but not by very much. A plausible explanation why `Eclipse`'s performance relative to BP seemingly *improved* with more time from 100 → 1200 sec (while BP usually improved with more time against alpha-beta programs in our other experiments) is that `Eclipse` is tuned for longer time tournaments. In particular, its transposition table and deep endgame search (features which our BP program lacks) become more effective at 300 sec than at 100 sec. To quote private communications from Giles & Springer:

> In anticipation of being able to do a relatively deep search, Eclipse stores on its first few iterations a *lot* of ordering information. When it's only got 100 seconds total, it probably never gets to the deeper searches where that extra ordering would pay off... In quick games (such as the ones you've been playing), my guess is that the transposition tables are actually slowing us down... Your results are showing Eclipse hitting its best area. Perhaps your idea that BP would gain ground on alpha-beta has merit, but the time controls need to increase a lot for BP to gain anything.

Still, we find this disturbing. It is also disturbing (and perhaps related!) that while `Eclipse` always consumed (on average) over 90% of its allotted time, BP consumed less and less time in longer games – thinking on average only 73% as long as `Eclipse` in the 20 minute games[21]. Probably this means that the vaunted time control algorithm of §4.5 is too simplified.

In most people's opinion `Logistello` is the world's strongest Othello entity. Buro ran BP vs `Logistello` tourneys on his SPARC-10/M20 with 64 MB. `Logistello` uses a negamaxing search with transposition table, Buro's "probcut" forward pruning mechanism, a 20+ ply endgame solver, and corner quiescence. Running at $\approx$ 72000 nodes/sec, it is 40 times faster than BP's 1350 evals/sec. At 240 sec/side/game, `Logistello` won 109 and BP won 25, with 8 draws (avg discs 36.28 to 27.68; time consumed 221 to 185). And in a time odds tourney (`Logistello` with 60 sec/game vs. BP with 240) `Logistello` won 79 and BP won 55, with 8 draws (avg discs 32.69 to 31.25; time consumed 57 vs 186). Buro estimated BP would draw even with `Logistello` at 6:1 time odds[22]. Considering the speed advantage of about a factor of 40 that `Logistello` has on our code, and that BP is using less of its time budget, at those 6:1 time odds BP would still be evaluating about 6-11$\times$ fewer positions.

Although this convincingly proves that `Logistello` is the stronger player, we are not discouraged. Buro's evaluation function is faster than ours by a factor of perhaps 25 with little loss of wisdom – a feat which we could presumably match with sufficient effort and intelligence. `Logistello` gains a speed factor of $\approx$ 2.2 from its transposition table, which gain might also be realizable in BP. BP could be improved with a `Logistello`-class endgame solver. There are various other possibilities for speeding our BP search code[23]. BP's strength seems to increase faster than AB's as the searches get larger, cf. §4.1.3. Finally, as shown by our very recent gain (equivalent to an effective time factor of perhaps 4) from improved parameter tuning, see §5.2, we are still early in the learning curve of engineering improvements, and expect other substantial gains to be discovered.

4.1.7 Time odds tourneys vs. brute force version of Logistello

Our experiments against our own alpha-beta Othello program Obogon reported in §4.1.3 suggested that our program Obippie increases its strength against alpha-beta the longer both programs have. We were unable to perform such experiments giving Obippie near tournament time limits (of 30 minutes) because Obogon would have had to have at least 100 times as much time, so that collection of tournament data became impractical. To test Obippie at longer time limits, we instead played it against a version of `Logistello`. To make the opponents more closely matched, and to simplify the interpretation of the results, we used a weakened version of `Logistello`, which has its "probcut" search turned off, and simply uses full-width negascout with iterative deepening, transposition table, and quiescence on corner moves. `Brutelog`'s evaluator differs drastically from the one in Obogon. We ran these tournaments from a 50-position starting set provided by

[21] Where BP consumed 690 sec in games it won, 792 in games it lost, and 716 in drawn games, on average. Presumably this significantly longer time consumption in lost games was caused by the loss, and not its cause (since clearly BP was rarely in time pressure).

[22] But later experiments (§4.1.7) suggest "6" should be closer to 20.

[23] Buro suggested switching to sub-single precision real numbers, in place of double precision, to save time and memory; another speedup might be to employ distribution compression within the search as in footnote 11 of [7]. On the other hand, Buro points out that 1. `Logistello`'s probcut search is most tuned for slower – 1800 sec/side – games; 2. on Buro's pentium, `logistello` gets 25K evals/sec while BP gets 1300, a ratio of 19, not 40; 3. building a `logistello`-class BP player might require BP to reduce its high memory consumption, which might introduce further difficulties, 4. transposition tables may not cause as great a speedup for BP as for AB.

Buro, rather than our usual 72-position Gatliff set, which Buro had criticized because 18 of these 72 Gatliff lines are in fact prefixes of other lines. The results are reported in table 12.

Brute sec	BP sec	Brute consumed	BP consumed	**ratio**	tourney result	%
30	60	16.9	46.5	2.75	43 (31.76) - 6 - 51 (32.08)	46.0
38	60	24.3	48.0	1.98	53 (33.46) - 10 - 37 (30.52)	58.0
45	60	31.6	47.9	1.52	58 (33.40) - 7 - 35 (30.58)	61.5
75	60	59.2	50.8	0.86	69 (36.16) - 6 - 25 (27.81)	72.0
40	120	26.0	100.5	3.87	44 (30.85) - 4 - 52 (33.15)	46.0
60	120	44.8	101.7	2.27	50 (32.23) - 9 - 41 (31.76)	54.5
75	120	59.4	99.9	1.68	69 (36.16) - 6 - 25 (27.81)	72.0
75	240	59.2	187.0	3.16	42 (30.41) - 9 - 49 (33.54)	46.5
100	240	83.6	195.9	2.34	38 (30.27) - 11 - 51 (33.68)	43.5
120	240	102.5	198.1	1.93	47 (31.20) - 6 - 47 (32.76)	50.0
150	240	131.5	203.2	1.55	54 (32.09) - 8 - 38 (31.89)	58.0
95	480	78.1	361.9	4.63	46 (31.42) - 5 - 49 (32.56)	48.5
120	480	102.0	379.1	3.72	52 (32.12) - 5 - 43 (31.85)	54.5
120	960	102.0	683.8	6.70	33 (30.99) - 10 - 39 (33.01)	46.3–
160	960	140.1	676.2	4.83	42 (31.70) - 11 - 47 (32.30)	47.5
240	960	216.5	716.4	3.31	50 (31.72) - 5 - 45 (32.27)	52.5 *
300	960	272.3	708.2	2.60	49 (31.80) - 9 - 42 (32.20)	53.5
240	1920	216.6	1188.2	5.49	39 (30.15) - 6 - 55 (33.84)	42.0
320	1920	290.3	1212.4	4.18	55 (32.60) - 3 - 42 (31.39)	56.5
480	1920	442.2	1211.8	2.74	55 (33.06) - 6 - 39 (30.94)	58.0
960	1920	879.7	1280.2	1.46	43 (33.52) - 5 - 18 (30.48)	68.9–
480	3840	449.5	2224.2	4.95	48 (31.93) - 6 - 46 (32.07)	51.0
640	3840	590.0	2191.7	3.71	52 (32.28) - 7 - 41 (31.72)	55.5
960	3840	879.7	2200.2	2.50	33 (32.80) - 8 - 24 (31.20)	56.9–

Table 12: Brute-force `Logistello` versus Obippie at various time odds. To explain the format by example: in the line ending "*": `Brutelog` was given 240 sec of thinking time per game (on average consuming 216.5), while BP had 960 sec (on average consuming 716.4). The ratio between those average consumed times was $3.31 \approx 716.4/216.5$. `Brutelog` won 50 games, lost 45, and drew 5 for a percentage score of 52.5% (draws count 1/2). However, `Brutelog` actually lost this tourney on average disc count (31.72 versus 32.27). This and the line after it were the only such anomalous tourneys. The lines ending "–" denote incomplete tourneys.

We see no evidence here that Obippie is increasing its strength relative to `Brutelog` in slower games; it may even be slightly *weakening*. Previously we had evidence that Obippie was weakening versus `Eclipse` in longer games (§4.1.6), but those results were not very statistically significant, Obippie's time usage had been decreasing versus `Eclipse` in longer games, and the authors of `Eclipse` had provided us with the convenient explanation that `Eclipse` was "hitting its tuned range."

We do not understand the disparity between the scaling of Obippie's performance relative to our alpha-beta program Obogon and the its scaling against `Brutelog`, but we provide the following conjectures.

1. Possibly the fact that Obippie and Obogon used the same evaluation function was favorable to Obippie. Perhaps taking the mean of Obippie's evaluator yields an evaluator with certain weaknesses exploitable by Obippie, or perhaps Obippie's evaluator has weaknesses that `Brutelog` but not Obogon are able to exploit.

2. Possibly there is a bug in Obippie, which becomes more visible in long games.

3. Possibly `Brutelog` and `Eclipse` had a higher quality alpha-beta search than Obogon. We think this unlikely. Node count tests indicate our move ordering is of comparable quality, and our evaluator seems at least equal theirs in same depth play.

4. `Brutelog` and `Eclipse` have transposition tables. Transposition tables are well known to be more important at deeper search, because of combinatorial effects on the numbers of transpositions[52]. Schaeffer [48] found that in chess as search depth increased from 3 to 9, transposition tables caused a speedup factor that increased roughly from 2 to 10. Negascout's benefit over plain AB also was an increasing (although small) function of depth in Schaeffer's experiments. However tournaments playing `Brutelog` against itself with trans table turned off (see table 14) do not show a dramatic increase in strength due to the trans table at larger game times.

5. `Brutelog` and `Eclipse` have special purpose endgame solvers which are allotted a substantial fraction ($> 50\%$) of the total time budget. We attempted to discover how important this was to the result by tabulating how many discs Obippie lost after `Brutelog`'s perfect solve. The results are shown in table 13. Unfortunately this doesn't tell the whole story. `Brutelog` features earlier win-loss-draw solves and bounds solves, which consume most of its running time, but unfortunately are not addressed at all by this data and presumably amplify it by a factor of ≈ 3, because there are about $3\times$ more moves available for Obippie to blunder on in the regime before `Brutelog` solves. The data does not seem good enough to rule for or against the "solver hypothesis."

6. M. Buro suggested Obippie might be falling into traps because it searches too selectively. If so, this could presumably be avoided by "safeguarding" the evaluation function by adjoining masses ϵ near $\pm\infty$ to the distribution. The value of ϵ best balancing safety and accuracy would have to be found empirically.

	tourney result						
BP time (sec)	60	120	240	480	960	1920	3840
Discs lost	.38	.42	.25	.25	.1	.28	.45

Table 13: Discs lost by Obippie after `Brutelog` had perfectly solved the endgame. Disc number is the mean of the two closest tournaments where Obippie had the indicated time.

sec	tourney result	%
30	52 (33.25) - 6 - 42 (30.73)	55.0
45	56 (32.80) - 7 - 37 (31.20)	59.5
60	52 (33.10) - 8 - 40 (30.89)	56.0
120	49 (31.99) - 11 - 40 (32.01)	54.5
300	53 (33.37) - 7 - 40 (30.62)	56.5
600	55 (33.56) - 8 - 37 (30.42)	59.0
1200	53 (33.37) - 7 - 40 (30.62)	56.5
2400	53 (32.91) - 11 - 36 (31.09)	58.5

Table 14: `Brutelog` versus itself with transposition tables turned off, at various time budgets. To explain the format by example: for the line beginning "120," both sides had 120 sec of thinking time per game. Brute (equipped with trans tables) won 49 games, lost 40, and drew 11 for a percentage score of 54.5% (draws count 1/2). The no trans table version actually won this particular tournament, however, if we go by mean disc count (32.01 versus 31.99). Trans tables speed up `Brutelog`'s search at tournament time scales by a factor ranging from $1.2\times$ to $3.3\times$ on 9 test positions.

4.2 Warri

Our latest BP Warri player is superior to our AB Warri player, see table 15.

Time	AB Wins	BP Wins	Draws	AB Seeds	BP Seeds	Seed Stddev	Conf.	**Seed Conf.**
80	136	196	48	23.37	24.63	4.97	2.18	2.48
160	131	199	50	23.20	24.80	4.09	2.47	3.82
240	134	207	39	23.16	24.84	4.34	2.65	3.76

Time	AB evals/game	BP evals/game	AB time used	BP time used
80	1236209 (±506316)	291333 (±100913)	75.34 (±29.18)	73.49 (±25.35)
160	2480482 (±990350)	562644 (±199473)	151.48 (±57.56)	147.02 (±53.04)
240	3758763 (±1549758)	834577 (±347939)	229.55 (±91.94)	221.22 (±92.84)

Table 15: Warri results at equal time usage.

At 80 sec/game, BP is stronger with 2.76σ worth of confidence, based on seed count (i.e. 99.7% confidence) and this advantage seems to increase in slower games. Notice that BP consumed slightly less time and had a node rate 4 times slower than AB's 16000 evals/sec.

The advantage of BP over AB is about as large as the advantage that AB would have gotten by searching 0.5-1.5 ply deeper (i.e. $\approx 2.25\times$ more time[24]) based on tournaments we played but omit.

Tree shape statistics are reported in table 16.

AB tree statistics (from the 80 sec/game tournament above):

Search depth:	12.03 (±3.58)
Number of leaves:	17107.11 (±27954.34)

BP tree statistics (averages over all search trees in all moves in all games)

Maximum leaf depth:	13.66 (±4.79)
Average leaf depth:	8.43 (±2.82)
Depth in true line:	7.78 (±3.84)
Number of leaves:	5584.18 (±7374.52)
Number of internal nodes:	1772.51 (±2321.71)

Table 16: Warri tree shape statistics. Mean (±Std. Deviation)

The 380 games were played starting from all 190 possible 3-ply Warri game starts. The evaluator had been constructed by our KS-tree style learner, this time based on data from positions arising in BP searches made by earlier BP programs. Both players were working under a chess-style time constraint of the following form: Allow each player T seconds for its first 40 moves, and $T/4$ more seconds for each 10 moves after that. BP is using a 10% gulp fraction.

It was suggested to us that perhaps BP's advantage was merely due to the AB player's lack of an end-off "quiescence," or capture, search. Since in Warri (like chess, but unlike Othello) there is a clear notion of "material" and of a "capture," it is obvious how to define such a search. But in a 1700-game tournament between an AB player searching to fixed depths 5-9, and an AB player searching to the same depth *plus* doing a further quiescence search on capturing moves, forced moves, moves by houses with ≥ 12 seeds, and moves where the nonmover had no seeds on his side (both sides used the BP-mean evaluation function and 10-seed

[24] AB's typical effective branching factor was 2.25 on average and 2.7 at game start.

endgame tables), the quiescence player, surprisingly, won slightly *fewer* seeds and games. Some hypotheses for this are: (a) our evaluator was sophisticated enough to already know much of what a quiescence search could tell it, (b) material is less important in Warri than in chess, (c) possibly the evaluator, which had been trained on positions from BP search trees, performs worse when applied to positions at the leaves of quiescence searches – a differently biased sort of statistical sample, cf. §6.1. We conclude that our AB player's lack of quiescence was not a problem for it.

4.2.1 Equal nodes tourneys

Since BP Othello was evaluating about 1350 nodes/sec as compared with AB's 2300, we did not see a large increase in BP's relative Othello performance under conditions of equal nodes (tourney results omitted in this paper) – about an extra .32 discs per game.

But our AB-BP Warri tournaments with an equal number of evaluations look rather good (see table 17) since they amount to cutting AB's time by a factor of 4.

Evals	AB Wins	BP Wins	Draws	AB Seeds	BP Seeds	Seed Stddev	Conf.	**Seed Conf.**
290,000	78	272	30	21.23	26.77	4.81	7.04	11.22
580,000	99	244	37	22.10	25.90	4.59	5.26	8.05
830,000	106	242	32	22.10	25.90	4.40	4.93	8.43

Evals	AB evals/game	BP evals/game	AB time used	BP time used
290,000	291467 (±128556)	296491 (±120885)	18.67 (±8.04)	72.57 (±29.00)
580,000	594003 (±246503)	573920 (±228036)	38.18 (±15.65)	146.38 (±59.50)
830,000	837706 (±335240)	834232 (±335095)	53.83 (±21.31)	218.19 (±88.43)

Table 17: Warri results at equal evals/game.

The number of nodes evaluated per game is not exactly equal, since we just tweaked the time parameters until the difference in evals/game between AB and BP was much less than 1 std. dev. of the evals/game made by AB.

4.3 Direct measurement of evaluation function impact

The experiment reported in this section unfortunately used a previous and less good Warri evaluator. We built AB and BP Warri players both using as evaluation function for a leaf, the result of a BP search to depth k on that leaf. (AB was given the mean of the distribution returned by the depth-k BP search.) By altering the value of k, we measure the effects of artificially changing the "brainpower" of the evaluator.

This sort of 2-stage tree search is also of interest (see part I) for the purpose of reducing memory consumption in BP search, and might be of interest if interfacing with special purpose hardware. In this experiment, we always used 3 spikes per (true) leaf and did not use the important trick of changing the number of spikes depending on depth. Also, we used an "eggtimer" time control, less favorable to BP (c.f. §4.1.4), in which each player must make each move in 2 seconds.

At $k = 0$, AB won 34, lost 27, and drew 11. We had 2.34σ of confidence (based on seed count) that it was better than BP. At $k = 1$, BP won 32, lost 26, and drew 14, with 2.22σ of confidence (based on seed count) that it was better than AB. At $k = 2$, BP won 43-21-8, with 1.93σ of confidence[25].

AB's superiority in the important case $k = 0$ probably stemmed from its 2.9 times faster node rate. However, as the evaluation function was made smarter (and slower) by increasing k, BP's win counts monotonically improved, until with $k = 2$, when the evaluation function is about 17 times slower than at $k = 0$, BP won games at a 2:1 ratio.

[25] This drop in reported confidence from the $k = 1$ case was due to a larger standard deviation in seed count.

This experiment supports our belief that BP gets stronger, relative to AB, when the evaluation function is slower and smarter. Slower evaluation function means BP has comparatively smaller overhead, and it is of course unfair to penalize AB for this. The following observation, however, provides one data point regarding the relative importance of "slower" and "smarter." The BP player here (with $k = 0$) was 2.9 times slower than AB, but lost, whereas our latest BP player, with smarter evaluator and better time control, is 4.0 times slower than (similarly smarter) AB, but wins.

4.4 Slagle Kalah

Even with our latest KS-tree (cf. §6.1.1,[54]) evaluation function for Slagle Kalah, BP with utility guided growth is unable to beat AB at equal time. The chess style time control in our tournament gave each player T seconds for their first 20 moves, plus $T/4$ seconds for every 5 additional moves. AB won 1009-711-202, with a confidence of superiority 4.81 σ.

The AB player was searching to an average depth of 30 – enough[26] to solve the game after not very many moves have gone by, and much deeper than our BP player (average leaf depth 9.2). Presumably this incredible depth is due to the speed of basic operations in Slagle kalah, the presence of a simple but effective move ordering (§7.3), and the presence of a large number of cutoffs from early win detections. Note that AB's node rate is about 8 times quicker than BP's, and all known evaluation functions for Slagle kalah are rather poor quality. All these conditions favor AB over BP.

BP is able to beat AB in Slagle kalah tournaments with equal numbers of *evaluations* by 15-44% more wins. With AB at depth 5 using 8% more evals than BP (using in fact an early, decision tree [but not our KS-tree], evaluator), BP still won 960-832-129, for 2.07σ of confidence. Since BP stores its search tree, one improvement in BP (which we have not yet implemented) is to retain the relevant portion of the stored tree to the next move. If one assumes this were implemented, it would be reasonable to only charge BP for *new* node expansions. BP won a tournament reflecting such scoring by 1043-726-137, or 5.13σ of confidence, in spite of AB using 10% more evals.

4.5 Time control mechanisms

We ended up using different time control mechanisms for BP Othello and BP Warri.

In neither case have we tried the suggestion made in §3.1 of [7], that the parameters in the time control be made explicitly dependent on the game stage. That idea might have allowed us to cure an annoying sickness in our present time control algorithms: they tend to consume substantially less than their allotted time in long games (§4.1.7).

4.5.1 Othello

Our Othello time control was based on "Szabo" version of a formula derived in §8 of [7]. Eqn 20 there estimates that we should stop searching and move when

$$\frac{U_{\text{gulp}}}{t_{\text{gulp}}} < c_4 (\frac{m}{t})^{1+c_5} \tag{5}$$

where t_{gulp} is the estimated time to do the next gulp, U_{gulp} is the utility estimate for that gulp, t is the time that would then remain to make the next m moves in, and c_4, and c_5 are positive real constants. In all of the experiments in this paper *except* for the 300 second match with `Eclipse` (§4.1.6) we took $c_5 = 0.076$ and then when times t are measured in seconds and utilities U are measured in discs, with our current hardware the best choice for c_4 appeared to be between 0.5 and 1.0. This suggested that $c_4 = 0.4\eta^{c_5}$, where η is the node rate (evals/sec), is a good initial try. (In Warri, perhaps $c_5 \approx 0.15$ would be more appropriate.) Later tuning experiments with 16 tournaments of 300 second games showed that a statistically significant two-dimensional maxima of score vs. `Eclipse` occurred near $c_5 = 0.09$ and $c_4 = 0.9$.

[26] Here searches which solved the game, or solved except for very few very deep lines, were counted as "depth 40," regardless of their actual depth. Really, among non-solving searches, the mean depth was closer to 18.

4.5.2 Warri

We found that with our current decision tree data, the BP Warri player believes that the utility of searching decreases as the game progresses. This may be due to a flaw in our evaluation function. Using our initial Warri time control (based on a previous hack not described in this paper), the BP player would devote most of its thinking time to the early moves and, more troublingly, it would refuse to think longer when we gave it more time. Therefore, we used the following time control whose main virtue is that the BP Warri player actually thinks for time proportional to the time limit specified. We have unfortunately not yet tried the more principled time control described in the previous subsection.

The Warri time control uses the following variables:

T_{tot} Time left on BP's clock.

M_{tot} Estimated number of moves left for BP to make.

T_{gulp} Estimated time that the next gulp will consume.

T_{used} Time used on the current move so far.

U_{gulp} Estimated utility of the next gulp.

U_{min} A lower bound on the allowed utility. This is added to U_{gulp}, because U_{gulp} sometimes drops to 0.

U_{final} An estimate of the utility that will remain in the search tree when we decide to move.

C_{w} Adjustable parameter. In latest experiments, set to 1.5.

The search is terminated if either of these conditions is true:

1. If $T_{\text{gulp}} > T_{\text{tot}}/2$, same as Othello.
2. If $C_{\text{w}}(U_{\text{gulp}} + U_{\text{min}})/U_{\text{final}} < (T_{\text{used}} + T_{\text{gulp}})M_{\text{tot}}/(T_{\text{tot}} - T_{\text{gulp}})$, or in words, if the ratio of estimated utility to final utility falls below the ratio of time spent on this move to time remaining per move.

One could estimate the U_{final} value in various ways. At present, we initialize U_{final} to a value somewhat lower than we expect it to be on the first move, based on a large number of games. After each BP move, it is updated to $U_{\text{final}} = 0.9U_{\text{final}} + 0.1U_{\text{gulp}}$. Thus if the utility drops for several moves in a row, U_{final} will drop and the BP player will adjust its utility meter to spend more time thinking.

5 Engineering tricks

Soon after the first alpha-beta chess players appeared, so did various engineering improvements upon them, including "quiescence" and [52] "iterative deepening." Although the rate has slowed, such improvements continue to appear even 40 years later [4, 11]. We similarly conjecture that there are many engineering tricks to be had in BP search.

5.1 The multispike trick

There is a tradeoff in BP between using staircase CDF's with many steps, which can approximate well arbitrary density functions, and using few steps, saving memory and time. In the initial growth stages, if one is using a 2-spike evaluator the search often finds its favorite move has an interval of support disjoint from those of the other moves. BP then concludes that there is "zero" probability that any further growth will change our opinion that this move is best, terminates the search early and makes the move. A better approximation of the true distributions would have indicated a small amount of overlap between the densities

in the moves. In the early stages, when the cost of another gulp is tiny, even a small overlap can motivate additional search.

The simple solution is to make the evaluator return different numbers of spikes at different search depths. At low depths, we return a 10-spike distribution. At high depths, we only return a 2-spike distribution. This costs hardly any time, since leaves at low depths are much less numerous. It also improves the play for a different reason: utility-guided tree growth decisions during the crucial early stages are guided by more accurate probabilistic information. This fix virtually eliminates the problem and vastly increases the strength of the BP player.

The same discreteness problem can occur less seriously at nodes deep in the tree – causing one to erroneously conclude that certain sibling leaves are "cut off" and have "zero" chance of being relevant. A better estimate of this tail probability would improve BP. A possible cure would be to revalue a set of siblings using a 3 spike distribution whenever our standard 2 spike distribution decides some are irrelevant. This would cost little time because we need not actually call the evaluator again – only to look up a precomputed 3-spike compressed distribution instead of the 2-spike one (having already got the bin indices and offsets on the previous call).

5.2 Tuning the gulp size

BP contains a parameter called "gulp fraction" that specifies the fraction of leaves expanded each gulp. Table 18 shows the results of tournaments between our BP Othello player with various gulp fractions, and our alpha-beta player.

Gulp size	AB sec used	BP sec used	Conf.	Disc Conf.
1%	85.22	58.75	4.57	7.84
2%	84.87	61.59	5.16	7.63
3%	85.58	64.71	5.06	7.51
4%	**85.68**	**69.05**	**5.52**	**10.45**
5%	84.60	73.83	4.45	8.28
6%	84.07	76.71	4.75	7.25
7%	85.16	80.07	3.26	6.07
10%	85.17	85.11	2.08	3.90
15%	84.44	85.80	0.36	0.70
20%	84.45	85.98	-0.95	-1.72

Table 18: BP with various gulp fractions versus fixed AB player.

The best choice seems to be about 4%.

The following model yields insight into the sensitive dependence on gulp fraction. Say we use gulp fraction f_1, but the fraction of leaves that are both in our gulp and "useful," is only f_2, $f_2 < f_1$. Then after s expansion steps ("gulps") the total number of leaves in the final tree will be g_1^s, where $g_1 = 1 + (b-1)f_1$ and b is the branching factor. However the total number of useful leaves, i.e. the number we would have had if we used gulp size f_2, is g_2^s, where $g_2 = 1 + (b-1)f_2$. We find that using f_2 as gulp size would have achieved the same information in a factor $(g_1/g_2)^s$ less time. Thus the saving caused by better selectivity, can grow *exponentially*.

This analysis and experience suggests that an engineering improvement along the lines suggested in §6.2 of [7] might pay substantial dividends. The idea there was to achieve greater selectivity by using a nontrivial notion of "expansion" of a leaf. For example one could order the leaves within a gulp, and as each leaf is expanded, approximate on the fly its children's expansion relevances, and expand immediately sufficiently important ones. We have not yet attempted such an improvement. These results also reinforce our intuition and tentative experience that extending BP to include partial node expansion may yield substantial payoffs.

Possibly a different gulp size might be preferable at longer time controls. The experiments reported in this paper were done using the 4% gulp size derived from tuning experiments at 100 seconds/side/game.

Notice also in table 18 that BP's time consumption decreased with gulp size. This was due to a flaw in our (then) time control algorithm. This caused us to discover and implement the simpler and better time control method, described in §4.5.1, used throughout this paper. Possibly we could further improve gulp tuning by redoing it using the new time control.

5.3 The large-utility trick

Occasionally single leaves occur with a sizable fraction (e.g. 20%) of the total importance in the gulp. These leaves mess up the gulp trick. The simple solution we implemented was to expand these leaves to depth 2 in a single gulp. This costs little, since there can be at most 5 such leaves. Perhaps we should have gone further in this direction as discussed in the previous section.

6 Learning methods to make evaluators

6.1 Linear regression and significance tests

Before constructing an evaluation function that returns probability distributions, we first construct a scalar evaluator, i.e. one that simply returns a number approximating the expected game result if play were to continue from a given position. Our scalar evaluators were constructed (after some experimentation) by piecewise multivariate linear regression. We devised a vector of "positional features." We found weights so that the weighted sum of the features was a least squares approximation to the final game result, over all positions arising in a very large database of games. Where it seemed to improve performance, we smoothed the boundaries between game stages[27].

We only accepted features that passed various tests of statistical significance. First, each feature's weight had to have absolute value at least 10 times its standard error. Second, features with small "importance" (weight times standard deviation) were suspect. Third, any features that come in matched pairs but behave unlike their complement aroused suspicion. Fourth, the fit had to show good prediction of game result (substantial decrease in residual). Finally, most features are expected by the human designer to have a weight of a certain sign and rough magnitude. Features not conforming to such expectations were suspect.

The procedure we adopted to design linearly-regressed evaluators was:

1. Design a set of features.
2. Do fits to game results in large games databases.
3. Do significance tests.
4. Based on results of step 3, redesign or delete suspicious features and/or add new ones.
5. Back to step 2 until fits are good and no suspicious features remain.

Our experience was that one needs 1000-3000 games per feature in order to get decent fits. Our Othello and checkers evaluators involve 30-60 features and are based on over 80000 games each. Our mod-9-connect-4 evaluator uses only 10 features.

M. Buro [11] presented evidence that logistic regression works better than plain linear regression for Othello evaluators. Our experiments (omitted) in chess have entirely supported that. Also in chess we found that fitting only "quiescent" positions was a superior procedure.

For Slagle kalah, which was written first, we used as scalar evaluator, a simple function (see footnote 3.3) instead of a fit to a number of sophisticated features.

[27] Occasionally weights were found to depend on ply number in an oscillatory manner. Othello expert and programmer David Parsons convinced us that such oscillations are often real. For that reason, when using weight-smoothing, we smoothed the even-numbered and odd-numbered ply separately.

In our later Warri program, we used a self-learned table-based evaluator (which may be thought of as a linear evaluator with about 30000 weights). See §6.1.3.

For details about our features and evaluation functions, see our long TR [54].

6.1.1 Kolmogorov-Smirnov decision trees

Once one has a good-quality scalar evaluator, one can semi-automatically construct an evaluator which returns a probability distribution. We call the method we invented "Kolmogorov-Smirnov trees."

First we acquire a large set of positions arising during BP searches. For each, we know a set of positional features, its scalar evaluator value, and its scalar evaluator value backed up by some number of plies of lookahead. The difference Δ between these two values is the "opinion change."

We then wish to develop, by learning from this dataset, an evaluator which will return the probability distribution of Δ conditioned upon the values of the features. The problem of learning to predict a probability distribution conditioned on features is also of great interest in applications entirely divorced from gameplaying, comprising a major subfield of statistics.

Our evaluators were "binary decision trees." Each node in such a tree is a yes-no question about a feature (or in principle a combination of features). One branches left or right according to the answer. At a leaf the remaining dataset is returned.

We built such decision trees by a greedy, recursive process. Start with a zero-node tree. Consider all possible inequivalent questions of the form "is feature i greater than x?" Each such question splits the dataset into two subsets. We regard each such subset as a (large sample from a) univariate probability density on Δ. Choose that question maximizing the confidence that its two induced probability distributions are *different*. This confidence is computed by means of the "Kolmogorov-Smirnov two sample test," [53, 31] applied to *uniquified*[28] data. We cease to split further when $(1-c)/s$ becomes smaller than some constant (we often used 0.001). Here c is the KS confidence that the two distributions really are different, and s is the number of candidate split-questions.

This procedure takes several hours for an $\approx 10^5$-point dataset in ≈ 30 dimensions, adequately fast for our purposes.

We expect better results could be obtained by continuing to split the tree until singleton datasets were obtained, and then pruning back according to a confidence criterion. Also we suspect a different tree growth procedure, based on information-theoretic entropy, might be better.

6.1.2 Compression of probability distributions

The distributions returned by our Kolmogorov-Smirnov tree evaluator typically contain a large number (≈ 200) of spikes. For BP we need distributions with 2-10 spikes. Thus we face the following compression problem, of interest in many applications besides gameplaying: compress a univariate probability distribution represented by N spikes, to a distribution with only k spikes approximating the original distribution "optimally".

We have relied on the following method. We choose the locations and heights of the k spikes so that the first $2k$ nontrivial *moments* of the two distributions agree. Such a compression exists and is unique, and may be found using a slick numerical method of Golub and Welsch [20].

This compression method suffers from at least two flaws. Firstly, if the N spikes happen to be grouped into less than k clusters, then the solution becomes very ill-conditioned, because the "extra" points can locate themselves near any of the clusters while still satisfying the moment equations to high accuracy. However, this appears to be the *only* source of numerical difficulties in the range ($k \leq 10$). The simple solution is not to use k values too large for numerical stability.

[28] Since the KS test is designed to be used for continuous distributions over the reals, bad results can be obtained if we base split decisions upon duplicated datapoints. Of course we later use the full dataset to fill the leaves (once the tree topology is determined) to avoid distorting the distributions.

Secondly, methods that preserve moments can exhibit problems when the data contains rare outliers since the high-order moments will be dominated by the outliers. In our datasets such problems do not seem to occur. (Avoiding this was a consideration in our choice of features.)

Other approximation criteria might be better than this moment based approach, and indeed WDS and Han La Poutre (Utrecht, Netherlands) have written a manuscript containing several dynamic programming algorithms that optimize various approximation criteria. This method was adequate for our purposes, and has the advantage that one can quickly update the dataset that is being compressed, as applied in §6.1.3.

For use with KS-tree based evaluators in BP, of course the needed compressed distributions are precomputed and stored in "bins;" the decision tree itself only stores bin indices in each leaf.

6.1.3 Learning as you play

We improved our evaluators by two different "learn as you play" methods. The first was to add new Δ datapoints to our bin data sets during play. This not only gains new data, but draws it from exactly the distribution encountered during BP play. This empirically turned out to be important for achieving strong BP performance. By only updating the *moments* of the bins (cf. §6.1.2) we can keep the memory usage bounded.

Tourney #	B wins	A wins	draw
0	27	31	4
1	29	26	7
2	33	22	7
3	35	22	5
4	35	20	7
5	36	19	7

Table 19: Learning in action: 6 consecutive tournaments between BP with utility-guided tree growth and an early learning decision tree evaluator ('B') versus an early, and nonlearning, minimaxing opponent ('A') in Slagle Kalah. 8-9 seconds thinking time per game.

The second was to use lookahead during play to gain data used to adjust the parameters in the scalar evaluator, in a Temporal-Difference like fashion. Details of both methods can be found in our TR[54].

We also used lookahead to learn scalar evaluators from *scratch* in Othello and Warri. In Othello we used an evaluator that added up a set of tabulated values, one table entry for every possible state of every possible line on the Othello board (there are 3^k states per k-long line, $k \leq 8$; we considered "bounce diagonals" to be "lines"), and with different sets of tables at 6 different game stages. Our evaluator tables in total constituted 708588 bytes. Because this evaluator is so simple, we obtained rates of 70000 evaluations/second during searches – enormous rates. Starting from a set of tables with all entries *zero*, we conducted learning negascout [37] searches with transposition table. After a few hours of learning, the program would make feeble attempts to avoid giving up a corner, but was still beaten soundly by a human beginner. After a week of learning, the program, called "`obaby`," had advanced to the point where it beat Colin Springer (1991 Canadian Othello champion) 2 games to 1 in a match, and achieved internet Othello server ratings > 1900. Considered as a feat of learning, this is superhuman. We defy any human to improve that much in Othello strength in any period even close to 1 week.

However, the resulting Othello program was nevertheless weak compared to our programs with handcoded features, hence was abandoned.

Our Warri evaluator was learned by similar techniques. Each side's eval was the sum of a "material score" and a "positional score," where the positional score was extracted from a table of most possible configurations of the stones on your side of the board. Our warri table learner had several advantages over our Othello table learner: The warri program was over twice as fast as the Othello program; the tables it needed to learn were over $10\times$ smaller; warri learning was started from a good evaluator instead of from all zeroes; and the warri learner had access to perfect endgame tables. Presumably due to these factors, the

resulting table-based evaluation function was very successful and was the one we chose for `w1`. (See TR[54] for details.)

7 Descriptions of the games

For a history of each game, a detailed description of features used in our evaluation functions, any new contributions we feel we have made in the study of that particular game, and a discussion of "the hall of fame" of the strongest gameplaying entities for each game and estimates of how our programs compare to them, see our lengthy Tech Report [54].

7.1 Othello

For the rules of Othello, see [26] or [40]. An important rule not mentioned by these sources is the scoring of games that terminate before the board is filled. In these games, *the winner gets the empties*. Thus a game ending with 21 white discs and 3 black ones would not be scored 21-3, but rather would be scored 61-3, and a 31-31 tie would be scored 32-32.

7.2 Warri

There are over 1000 members of the family of mancala pit and pebble games. Many of them are listed in [41] and [17]. The most important of the rule variants, and the one that is adopted in Antiguan league play (and in the annual tournaments held there in Decembers and televised in recent years) is called Warri. These are extracted from pages 15-17 of [41] and from [13].

1. Warri is played on a 2×6 board.
2. Four seeds per hole at gamestart (i.e. 48 total). South moves first.
3. To move: remove the seeds from a nonempty hole on your side of the board and sow (that is deposit seeds one by one in successive holes anticlockwise) until exhausted.
4. Except that the selected (source) hole is skipped over during sowing, so it will always be empty after the move is complete, even if the sowing went completely around the board for 1 or more cycles.
5. If last seed sowed lands in opponent's hole and makes a count of 2 or 3 seeds in that hole, then these seeds are removed and kept by capturing player in his "treasury," as are any seeds in any unbroken sequence of the opponent's holes, each containing 2 or 3, immediately preceding this hole.
6. If all your opponent's holes are empty, you must make a move (if one exists) that moves seeds into them, however briefly (conceivably you'd capture some or all of the men you moved, which would still be a legal move). If no such move exists, rule 7 will apply next turn...
7. If all your (i.e. the mover's) holes are empty, the game is over, and all remaining seeds go to your opponent's treasury.
8. The object is to capture the most seeds. You win if you capture > 24 seeds. Drawn games can occur (24-24). It is also possible via "perpetual cycles" for neither player to have > 24 and where, with optimal play, no more captures can occur. In this case (detected by a 3-time repetition) the simplest scoring method is to divide the cycling seeds evenly between the players, so that whoever was ahead before the cycle started, wins.

7.3 Slagle Kalah

Slagle kalah was introduced in papers by Slagle et al. [50, 51], who used it as a vehicle for studying game tree search, and studied by other AI researchers (see eg [15]). See [50, 51] for the rules. We call this game "Slagle Kalah" because, as far as we are able to determine, the particular Mancala rule variant used here was invented by Slagle. This game is quite simple, and in fact our latest software and hardware can solve it in about ten minutes[29].

7.4 Mod-9 connect 4

Mod-9-Connect-4 is played on a 9×9 board with horizontal cylindrical wraparound. Players move alternately. On each move, the player selects one of the 9 columns of the board and places a disk of his color on the lowest unoccupied square on that column. You win if you get 4 in a row horizontally, vertically, or diagonally. Draws can occur if the board fills up, but they are very rare. This game is similar to the game sold by Milton-Bradley and played on a 6×7 noncylindrical board, but that game has been solved (win for the first player by moving into the center column) by James Allen and L.V. Allis in 1989 [3, 55]. The present game was intentionally made larger and the columns were given an odd height (Allis's solver utilized various theorems about connect-4 variants with even-height columns), in an effort to make the game intractable.

8 Discussion

In our experiments, BP performed well. BP beat our alpha-beta programs at large time odds in Othello, and beat our alpha-beta programs in Warri. BP's advantage over alpha-beta seemed to grow as the game became more complex (Kalah→Warri→Othello) and as the time controls became longer. BP, although performing creditably, did not display the same gains in performance with longer time controls against two other alpha-beta programs, but these experiments were clearly not on an even playing field because these programs incorporated many features that Obippie did not, but presumably could have, and these programs also used very different evaluation functions.

We believe that BP will improve relative to AB when the evaluator is made more accurate. This is intuitively plausible: BP is using the evaluator both to valuate the tree and to shape it. This conclusion is supported both by the results of this paper (see especially §4.3), and also by our general experience building BP programs: the more knowledgeable and statistically accurate our evaluators, and the better statistical tools we built for constructing evaluators, the better our BP programs performed. Although it has proven possible to construct AB evaluators entirely by means of intuition and guesswork, we suspect that BP evaluators will always need to be built with the aid of statistical tools. We also conjecture that BP will do well in games where "accumulation of small advantages" is possible, because AB has trouble distinguishing between two candidate moves whose distributions are highly overlapping peaks, but BP simply selects the one with greater mean.

We would like to conjecture, tentatively, that BP with utility guided growth and a sufficiently good evaluation function, thinking for time t would play at equal strength to AB thinking for time $t \cdot g(t)$, where the "giveaway factor" $g(t)$ is given by a power law: $g(t) \propto t^P$ for some constant $P > 0$. Or in other words, BP search with utility-guided tree growth would be asymptotically about as powerful as plain AB search going a constant factor deeper than it normally would in time t.

The only experimental support for this law so far has been the 4 points in the loglog plot of figure 1 – our law would predict that this plot should asymptotically look like a straight line. For those four points AB needs to go to depth $\approx 2.5d - 7.7$ instead of its normal depth d to equal BP. It would be nice if we had

[29] The first player wins by playing '5.' The go-again move '4 5' draws and all other first moves lose. We have also solved the larger version with 4 seeds per hole at gamestart. The first solution of Slagle kalah was by Igor Rivin using a modification of our AB program. Endgame tables speed up the solve and the following move ordering is extremely effective: First go-again moves, then captures, then noncaptures, breaking ties so that most-forward source holes are considered first.

the data to extrapolate this further– 4 points, the last of which is speculative, don't provide it; and later experiments playing BP versus a version of `Logistello` (§4.1.7) did not seem to support this law.

Here is some theoretical intuition about why this law may hold. Some decades ago R.W. Floyd suggested a model of search in which the children of a tree node have positive real "plausibility values" (summing to 1) corresponding to the probability that they are the right move. The product of the plausibilities along a root-node path is the "plausibility" of that whole line of play, if we pretend these plausibilities are independent. Suppose the "right thing" for a tree searcher to do is *not* to go down to constant depth, but instead to go down all lines of play until their plausibilities sink below some threshhold. Floyd proposed making and using an a priori heuristic plausibility function for this purpose. If c, $0 < c < 1$, is any constant, and assuming the branching factor is ≥ 2, then the N^c deepest (among N total) Floyd lines will penetrate a constant factor deeper than average, with probability$\rightarrow 1$. Presumably one cannot afford to mis-estimate this many lines without risking making the wrong move – leading to the desired law.

From the point of view of the programmer, BP and alpha-beta programs have some differences. A crude program based on BP with utility guided tree growth is more difficult to implement than a crude alpha-beta gameplayer, because the search algorithms are more complicated, and because it is necessary to write statistical evaluator-building tools. On the other hand, once your program is running, the task of gradually increasing its strength may actually be easier for the BP program. In an alpha-beta program, you need to work on the evaluator, search heuristics, and time control heuristics. Top alpha-beta chess programs have many search extension heuristics, and the interplay among them, and between them and the evaluator is mysterious. If one changes a term in the evaluator, it might in principle change the best choice of search extension heuristics. Keeping up with these effects is costly. BP takes care of shaping the search and dealing with time control issues, so you don't have to. In BP you can focus on developing the evaluator.

Obvious things to try next might include:

- Transposition tables – how should they best be implemented in BP, how should one best handle the issues related to BP in DAGs (cf. [54]), and how does all this affect performance?
- 2-stage BP search to reduce memory consumption (cf [54]).
- Distribution compression within the BP search to reduce time and space needs at the sacrifice of some accuracy.
- Variants of BP with partial node expansion, or anyway different kinds of "expansion," for example deeper, guided expansion of very high utility nodes.
- Further investigation of automated statistical methods for generating evaluation functions for BP.
- "Safeguarding" BP as suggested in §4.1.7, item 6.
- Better time control algorithms.

And then, one might want to try writing a BP chess program, which will involve extensive engineering to achieve high quality distribution valued evaluation functions.

In yet another direction, we wonder if BP search might have an impact in 1-player "games," such as the traveling salesman problem.

Acknowledgements: The following people helped us by providing some combination of computer code, information, data, ideas, or questions, and/or by virtue of being expert human gameplayers who played our programs. They are listed in roughly decreasing order of importance:

Michael Buro (Paderborn, Germany); Jonathan Schaeffer (Edmonton, Canada); Jean-Christophe Weill (Paris, France); Colin Springer (Minnesota); Mike Giles (Detroit MI); David B. Chamberlin and Mark Masten (Millersville PA); Henry Cejtin (NECI, Princeton NJ); Robert Gatliff (`io.com`); David Parsons (New York NY); Stuart Russell (Berkeley CA).

We would also like to thank Daniel Sleator et al (Pittsburgh PA) and Igor Durdanovic (Paderborn, Germany) for writing the internet chess and Othello servers, respectively, which are invaluable tools for anybody engaged in computer research on these games, as well as being a great source of entertainment.

References

[1] Foreman S. Acton: Numerical methods that work, MAA 1990 (updated from 1970 edition).

[2] Alan Agresti: Categorical data analysis, Wiley 1990

[3] Louis Victor Allis: Searching for solutions in games and artificial intelligence, CIP-Gegevens Koninklijke Bibliotheek, Den Haag 1994; ISBN=90-9007488-0

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

[5] 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

[6] Thomas S. Anantharaman: Extension heuristics, ICCA Journal 14,2 (June 1991) 47-65.

[7] Eric B. Baum and Warren D. Smith: Best Play for Imperfect Players and Game Tree Search; part I - theory.

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

[9] H.J.Berliner & C.McConnell: B* probability-based search, Artificial Intelligence 86,1 (1996) 97-156

[10] Leo Breiman, J.H. Friedman, R.A.Olshen, C.J. Stone: Classification and regression trees, Wadsworth 1984

[11] Michael Buro: Techniken für die Bewertung von Spielsituationen anhand von Beispielen, Ph.D thesis, at University of Paderborn, Germany, December 1994.

[12] Michael Buro: ProbCut: an effective selective extension of the $\alpha\beta$ algorithm, ICCA Journal 18,2 (1995) 71-76.

[13] David B. Chamberlin: How to play Warri, privately printed 1984. (Available from author, 2101 Birchwood Road, Lancaster PA 17603, for $7.)

[14] I. Chernev: The compleat Draughts player, Oxford University Press 1981.

[15] P-C. Chi & 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) pp 451-471.

[16] A. Delcher, S. Kasif: Improved Decision Making in Game Trees: Recovering from Pathology, Proceedings of the National Conference on Artificial Intelligence (July 1992) 513-518.

[17] A.Deledicq and A.Popova: Wari et Solo, le jeu de calculs Africain, CEDIC (93 avenue d'Italie 75013 Paris) 1977

[18] G.Goetsch & M.S¿ Campbell: Experiments with the null-move heuristic, in T.A.Marsland & J.Schaeffer eds., Computers, chess and cognitions, Springer 1990.

[19] R.M.Goodman and P. Smyth: Decision tree design from a communication theory standpoint, IEEE Trans. Info. Theory 34,5 (1988) 979-994.

[20] G.H. Golub and J.H. Welsch: Calculation of Gauss quadrature rules, Math. of Computation 23 (1969) 221-230 and microfiche.

[21] R. Floyd and R. Rivest: Expected time bounds for selection, Commun. ACM 18,3 (March 1975) 165-173

[22] Louis C. Ginsberg: Principles of strategy in the game of checkers, privately printed 1931. Reprinted by Don Goodwin, 51 Tefly Road, Willowdale, Ontario Canada M2M-1C5.

[23] E.T.Jaynes: Concentration of distributions, pp 315-336 in E.T. Jaynes: papers on probability, statistics, and statistical physics, Kluwer 1989.

[24] Robert L. Jennrich: Stepwise regression, pp. 58-75 in: Statistical Methods for Digital Computers, (Editors: Kurt Enslein, Anthony Ralston, Herbert S. Wilf) Wiley 1977

[25] Alexander Kotov: Think like a grandmaster, Batsford 1971

[26] Ted Landau: Othello, brief and basic (1984), sold by US Othello Association, 920 Northgate Ave. Waynesboro VA 22980-3425.

[27] Han La Poutre and Warren D. Smith: Approximation of staircases by staircases, Technical report, NECI, 4 Independence Way, Princeton NJ 08540.

[28] Kai-Fu Lee and Sanjoy Mahajan: The development of a world class Othello program, Artificial Intelligence 43 (1990) 21-36

[29] R. Levinson & R. Snyder: DISTANCE: Toward the unification of chess knowledge, ICCA (Int'l Computer Chess Assoc.) Journal 16,3 (Sept. 1993) 123-136.

[30] T.A. Marsland: A review of game tree pruning, ICCA Journal 9,1 (March 1986) 3-19

[31] F.J. Massey: Distribution table for the deviation between two sample cumulatives, Ann. Math. Statist. 23 (1952) 435-441.

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

[33] Dana S. Nau: Pathology on game trees revisited, and an alternative to minimaxing, AI 21 (1983) 224-244.

[34] A.J. Palay: Searching with probabilities, Pitman 1985

[35] Judea Pearl: Heuristics, Addison-Wesley 1985.

[36] J. Ross Quinlan and R. L. Rivest: Inferring Decision Trees Using the Minimum Description Length Principle, Information and Computation 80,3 (March 1989), 227-248.

[37] A. Reinefeld: An improvement of the scout tree search algorithm, ICCA Journal 6,4 (Dec 1983) 4-14

[38] Arthur Reisman: Checkers made easy, Key publ. co. 1959

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

[40] Paul S. Rosenbloom: A world-championship level Othello program, Artificial Intelligence 19 (1982) 279-320

[41] Laurence Russ: Mancala Games, Reference Publications Inc (218 St. Clair River Drive, Box 344, Algonac MI 48001) 1984

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

[43] S. Russell, personal communication.

[44] A.L. Samuel: Some studies in machine learning using the game of checkers, IBM J. Res. & Devel. 3,3 (1959) 210-229.

[45] A.L. Samuel: Some studies in machine learning using the game of checkers II – recent progress, IBM J. Res. & Devel. 11,6 (1967) 601-617.

[46] J. Schaeffer: Conspiracy numbers, Artificial Intelligence 43 (1990) 67-84

[47] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, D. Szafron: A world championship calibre checkers program, Artificial Intelligence 53 (1992) 273-289.

[48] J. Schaeffer: Experiments in search and knowledge, TR 86-12, Department of Computer Science, University of Alberta, Edmonton, Alberta, Canada. (His PhD thesis from U. Waterloo, May 1986.) See also his later summary in IEEE Transactions on pattern analysis and machine intelligence 11,11 (1989) 1203-1212.

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

[50] J.R. Slagle and J.K. Dixon: Experiments with some programs that search game trees, Commun. ACM 16,2 (1969) 189-207

[51] J.R. Slagle and J.K. Dixon: Experiments with the M & N tree searching program, Commun. ACM 13,3 (March 1970) 147-153

[52] D.J. Slate & L.R.Atkin: Chess 4.5: The Northwestern University chess program, in P.Frey (ed.) Chess skill in man and machine, Springer-Verlag 1983

[53] N. Smirnov: Tables for estimating the goodness of fit of empirical distributions, Annals Math. Statist. 19 (1948) 280-281

[54] Smith, W. D., E. B. Baum, C. Garrett, R. Tudor:
Best Play for Imperfect Players and Game Tree Search; part II- experiments; Monster Unedited Version; http://www.neci.nj.nec.com:80/homepages/eric/monster.ps.

[55] J.W. Uiterwijk, J.J. van den Herik, L.V. Allis: A knowledge-based approach to connect-four, in: Heuristic programming and artificial intelligence, the first computer olympiad, Ellis Horwood Ltd 1989

[56] S.H. Walker & D.B. Duncan: Estimation of the probability of an event as a function of several independent variables, Biometrika 54 (1967) 167-179.

[57] J-C. Weill: The NegaC* search, ICCA Journal 15,1 (March 1992) 3-7

[58] Tom Wiswell: The science of checkers and draughts, A.S. Barnes 1973.

[59] Tom Wiswell: The complete guide to checkers, Macmillan 1970

[60] Tom Wiswell and Jules Leopold: The wonderful world of checkers and draughts, A.S. Barnes 1980.
****

[61] Brian W. Kernighan, Rob Pike: The UNIX programming environment, Prentice-Hall, Englewood Cliffs NJ 1984

[62] Cleve B. Moler: MATLAB User's Guide, The MathWorks, Inc. Cochituate Place 24 Prime Park Way Natick, MA 01760.

[63] John K. Ousterhout: Tcl and the Tk toolkit, Addison-Wesley, Reading MA 1994

[64] Larry Wall and Randal L. Schwartz: Programming perl, O'Reilly & Associates, Sebastopol CA 1990

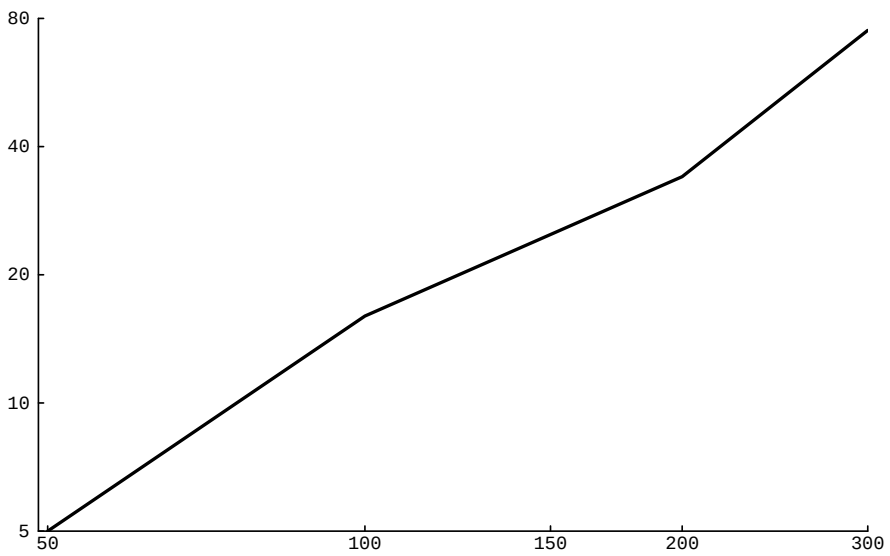

Figure 1: Loglog plot of allowable BP-AB time consumption giveaway factor (vertical axis: our best guess, based on time odds tourney table 3 ; readers may conjure up their own error bars...) in Othello, versus BP thinking time allowance per game (seconds; BP evaluated 1350 nodes/sec as compared with AB's 2300).

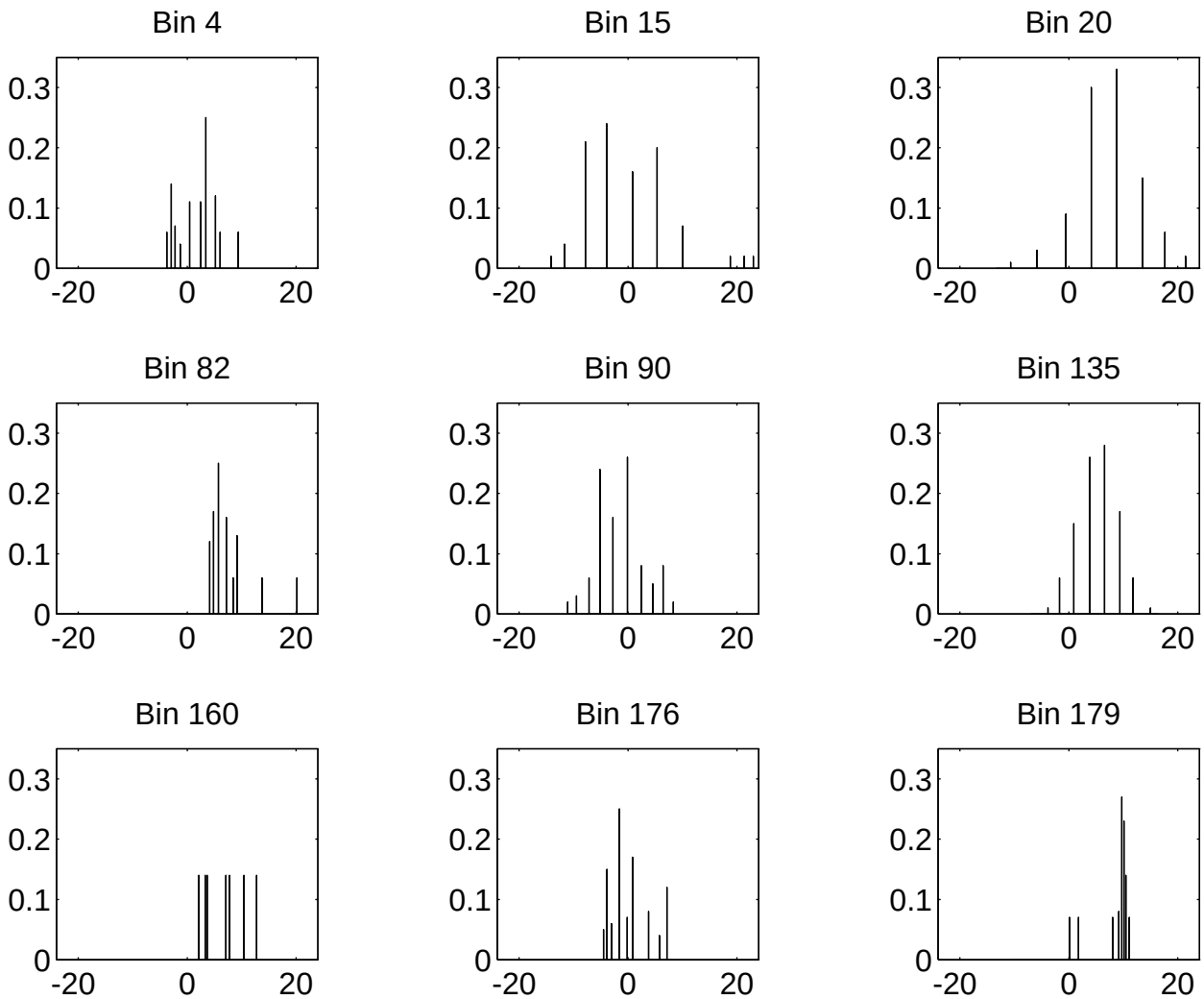

Figure 2: Pictures of 9 distributions from our Othello player's KS tree bins, produced by moment based compression of opinion change data at depth 5-6.