# Best Play for Imperfect Players and Game Tree Search; part II - experiments

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

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

DRAFT

September 19, 1995

**Abstract.**

In part 1 we proposed the BPIP algorithm for game playing. BPIP 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 accounting in a detailed way for the relevance of each leaf. Here we describe the first implementation of this algorithm and report on experiments comparing it with the standard (alpha-beta, or AB) approach, and the "naive probability update" (NPU) approach, in several popular games. BPIP is seen to evaluate a fixed tree more accurately than either AB or NPU in a large variety of games. BPIP 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 have invented several important BPIP-gameplayer engineering tricks in the course of this research, specifically, the "multispike trick," and new methods of evaluation function design. More such tricks undoubtably remain to be discovered.

# 1 Introduction

In a companion paper ("Part I - theory"), we proposed the "BPIP-DFISA" procedure for making computers play games such as chess. The BPIP approach both values a given search tree differently and grows a more selective tree than standard approaches. Part 1 argued that our procedure was optimal within certain approximations. This paper reports on experimental tests of these approximations in a wide variety of games, and finds in some cases substantial deviations. Nonetheless, we find that BPIP is able to best strong alpha-beta programs at Othello and Warri. We report as well ongoing research in discovering engineering tricks to improve BPIP's play and in training evaluation functions. BPIP's advantage appears to grow with the complexity of the search (e.g. the amount of time alloted the players or the complexity of the game). These trends may portend substantial edges for Bayesian search techniques, which attempt to explore the most relevant directions, as opposed to simple Branch and Bound in a variety of problems.

Shannon[47] 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 procedure speeds up this search. Various heuristic improvements such as "move ordering", "iterative deepening" and "killer tables" 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 certain lines judged important. 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]

The BPIP approach maintains a model of its uncertainty and grows and values the tree appropriately. Its evaluation function assigns to each leaf a probability distribution representing the probabilities this leaf would be found to have various values if searched more deeply. The standard, numerical evaluation function corresponds to the mean of our distribution valued evaluation function, but our distribution valued function contains more information about uncertainties. With the hypothesis that the distributions at different leaves are independent, BPIP evaluates any given partial tree to find distributions at each node expressing the probability that node would be found to have different values if expanded further. Given a partial tree, the best move is the one whose distribution has highest mean. From this point of view, the standard (Shannon) approach takes means at the leaves and propagates these mean values by minimax, whereas we propagate the full distributions and only take the means at the point of decision. Thus our approach accounts for the relevance of every feature in the distribution at each leaf in the context of all the other leaf distributions. If the independence assumption were fully valid, our approach would clearly be more accurate. We have experimentally examined the probability distributions, and report substantial deviations from independence. Nonetheless, our experiments show that BPIP more accurately values full width trees in all the games we tried, at every[2] depth of tree we tried. In some games, BPIP's advantage over minimax is comparable to that gained by an additional ply of search.

We also compare both alpha-beta and BPIP to a previously proposed probablistic scheme that we call Naive Probability Update or NPU. By NPU we mean the proposal[33] to use an evaluation function estimating probability of winning, and to compute the value of a node as the probability it is winning given the (assumed independent) estimates for its children. We predicted in part 1 that because NPU assumes the estimates themselves are independent, whereas BPIP merely assumes our errors in the estimates are independent, NPU would have far more serious problems with probabilistic correlations. We report that such problems do indeed arise in our experiments.

The BPIP approach also tries to grow a near optimal search tree. Given the model of uncertainty which it maintains, there is a formally well defined "most relevant" or "best" leaf to examine next, in the sense that if you were going to be able to expand a fixed number of further leaves, this best one would give greatest probability of winning. Unfortunately, computing which leaf this is is intractable. Instead we proposed in

[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 13.

part 1 a procedure for efficiently estimating the utility of each leaf and expanding a set of the most relevant ones. This procedure made several approximations, including statistical independence assumptions, and "the Gulp trick", which was intuitively reasonable, and necessary for the efficiency of the algorithm, but which involves a mathematically uncontrolled approximation. To examine whether this procedure generates accurate enough estimates we played games against alpha-beta where each algorithm evaluated the same number of nodes. BPIP beat alpha-beta at every game tried by this measure.

Another interesting question is what shape tree BPIP will grow. This depends on where BPIP thinks the most relevant leaves are as it expands. Experimentally BPIP grows a more focussed tree than alpha-beta, but by no means as focussed as human game players.

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]. This raises the ultimate question: Can BPIP beat alpha-beta in a fair game, with each given equal time. We found that BPIP beats alpha-beta decisively in Othello and in Warri – in the case of Othello, even when the competing alpha-beta program is given sizeable time odds.

Our goal in this research has been to understand the capabilities of BPIP, 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 BPIP's value as a search strategy. We have not yet invested in such improvements.

In many of our experiments, our BPIP 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 BPIP function. Our alpha-beta programs employed sophisticated move ordering heuristics. There are a set of known tricks which 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"[11] in the game of Othello, we did implement this (§4.1.5, 4.2.1 and tested BPIP 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 BPIP competitors have played even more strongly against open competition. We have also played several matches between our BPIP 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 BPIP program appears to be almost as strong as the second best program in the world, and substantially stronger than the third.

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[37]. Schaeffer [44] implemented the conspiracy number algorithm of McAllester[30] 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[32]. Russell and Wefald reported beating alpha-beta at Othello[40], but their implementation of alpha beta used no move ordering heuristic[41]. Our results are the first of which we are aware where an alternative approach was able to beat alpha-beta programs incorporating move ordering heuristics. This previous work and some additional is discussed in more depth in Part 1.

The original alpha-beta proposal has been strengthened by 40 years of engineering improvements. Achieving our results has required several engineering improvements such as the "multispike trick", which allows us to maintain adequately detailed distributions at all levels of our tree without unduly increasing our computational overhead. We also have had to develop methods for training up our distribution valued evaluation functions. We believe that such engineering tricks and learning methods are in their infancy, and that our

[3]In footnote 21 of part I we showed how this logarithmic factor may be gotten rid of at the cost of making the BPIP valuation less "exact." We have not experimented with that idea.

current program is thus subject to much improvement. Because BPIP 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 section 5 and our learning methods are described in section 6.

This paper is a brief summary aimed at a general audience. We hope to provide enough detail to convince the reader that our experiments fairly test BPIP, and to illuminate BPIP's strengths and weaknesses. We hope to provide some insight into the engineering work that has gone into our current implementations and what avenues might lead to further improvement. The reader interested in replicating or extending our results will find gritty details of our experiments in our (completely unpolished) Technical report [52].

# 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 [48], Othello [24], and Warri [12]. 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. 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 [33] [31]. 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` [59] and `C++` [63],although we have also used the following languages in various places: UNIX(TM) shell, `sed`, and `awk` [60], `TCL` [62], `perl` [64], and `matlab` [61].

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 Opponents

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

Our own alpha-beta opponents use good move-ordering heuristics to shape and grow the tree, and use "iterative deepening"[50] for time control and move ordering. Occasionally other tricks were used. For example we implemented quiesence where we felt it would improve alpha beta's play. 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[35], buy little advantage in practice. See[46] for a comparative study of such modifications. We will describe in section 6 how we trained up evaluation functions. Generally our alpha-beta Program used as evaluation function the mean of the BPIP distributions. This allows a direct comparison.

With an Internet Othello Server (IOS) rating 2039, our alpha-beta Othello program Obogon was ranked higher than any human on the IOS. We also report below on matches against the top three ranked 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 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 [10, 55] find game theoretic value with 24 empty squares.)

Logistello searches about 25 times as many nodes as Obogon. The biggest reason for our slow speed was our desire to keep the code comparatively simple and understandable. We believe our Othello evaluator is quite accurate.

The fraction of time BPIP 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 BPIP program Obippie would still dominate Obogon, but its edge would be smaller.

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 master 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[4]. However, we have never verified these conjectures.

The AB Warri program that we used for our BPIP vs AB experiments is based on a simplified version of `w1` which 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 quiesence search but were unable to design one which strengthened the play of the program.

Neither our BPIP nor our AB players utilized transposition tables[5] or "multistage evaluators"[6]. We conjecture that both sides suffer equally from these omissions, but this is a subject for future research.

AB used, but BPIP was denied, partial node expansion aided by a heuristic move ordering. Some preliminary experiments in Othello suggest that this might boost BPIP's strength substantially.

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

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

[6] Multistage or 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 BPIP search in various ways (some were mentioned in Part I).

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). These things are difficult to program well, and often do not buy very much improvement[7].

### 2.4 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 $\Delta$. 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 $\leq 1$ one would conclude that "A is stronger than B with confidence worth at least

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

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 can be obtained with smaller tournaments 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 between players of this type, let $\Delta$ be the difference in sums of the final scores of players A and B over a $2N$-game color balanced tourney, and let $\sigma$ 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

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

normal standard deviations."

One could alternatively assume that all $N$ game *pairs* were independent. The formula would then be $\approx \frac{1}{\sigma_2}\Delta/\sqrt{\mathrm{N}}$ with $\sigma_2$, 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 $\sigma_2$ and hence the former estimate (EQ 2) is more conservative than the latter one. We have used the more conservative (EQ 2) throughout this paper.

# 3 Experiments, stage 1 – Comparison of NPU, AB, and BPIP as statistical decision procedures

We first compared BPIP, minimax (AB), and Naive Probability Update ("NPU") 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. NPU, BPIP, and AB play identically at depth 1 throughout.

[7]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 BPIP enjoyed over AB in our Othello experiments, but comparable to BPIP's advantage in our Warri experiments.

## 3.1 Pearl's "P-game"

This game was designed by Pearl[33] and studied by Nau[31] as an example of a theoretically "pathological" game[8], i.e. a game where searching deeper can be shown to give smaller probability of making the correct move in some positions. 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 $n$ 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 $e2(n) \equiv r^{[h]}(f)$. Here the superscript denotes functional iteration. $e2(n)$ is the probability that node $n$ is a perfect-play win, given that its leaf descendants are 1 with probability $f$.

Recall that the BPIP distribution measures the likelihood of "opinion changes" as a node is expanded "further". To approximate BPIP, we must choose a definition of how much further to expand in producing our evaluation distributions. In the limit we expand depth 0, the BPIP distribution is a single spike and BPIP is identical to minimax. In the limit where the node is expanded to infinite depth, BPIP is identical to NPU. We estimated the distribution assuming expansion of depth 1. Thus we used for BPIP a two spike distribution, one spike assuming the $f$ given for $n$ 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,NPU, and BPIP players on 100,000 P-games, for[9] $k = 2, ...9$. 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 |
| " " NPU | 199926 | 197412 | 203727 | 202399 | 211349 | 207292 | 211538 | 203939 |
| " " BPIP | 203896 | 205825 | 211726 | 210619 | 216580 | 211967 | 215647 | 208904 |

This confirms Nau's [31] results that NPU is a superior decision procedure to minimax, searching to fixed depth in the P-game. BPIP is found to be superior to NPU with 4-9 standard deviations of confidence, depending on which depth. The advantages are small in an absolute sense.

## 3.2 Mod-9 connect 4

Mod-9 connect 4 is a larger version of the standard connect 4, with horizontal wraparound. We played AB vs. NPU from all possible inequivalent 3-ply starts. We did not design a BPIP 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.

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

For calibration, we played alpha beta vs alpha beta at increased depth.

| 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 |

[8] AB with 1, 2, or 3 extra plies bested AB in our tournaments, but AB variants which 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 NPU.

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

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

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

### 3.3 Slagle kalah

We played NPU against AB where both sides used Henry Cejtin's simple probabilistic evaluator[10] 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
" "     NPU    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 NPU wins at depths 10-12.

NPU 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 [14] showed that NPU was superior to AB at certain search depths when a reduced version of Slagle Kalah. They argued that NPU 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 which 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 BPIP's evaluation function. The results are in table 1.

| Depth | AB wins | BPIP wins | Draws | **Conf.** | AB wins | BPIP wins | Draws | Conf. |
|---|---|---|---|---|---|---|---|---|
| 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, ($\geq 3$) 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).

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

```
        depth   2   3   4   5   6   7   8   9   10  total
```

[10]This evaluation is the exact probability of winning given the current score difference (what [48] 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.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

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

BPIP and NPU do not use directly comparable evaluation functions. We played a tournament using different evaluation functions[11] 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. Totalling depths 2-7: BPIP won 218, NPU won 156, with 23 draws.

## 3.4 Othello

| Depth | AB Wins | BPIP Wins | Draws | AB mean Discs | BPIP " | 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 BPIP at depth 2, it was only by $0.61\sigma$ based on disc count (i.e. 73% confidence), and even less based on win count. The BPIP 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, BPIP has the advantage, with respectively $1.68\sigma$ and $2.36\sigma$ (i.e. 95% and 99% confidence) based on disc count.

Note that both here and in the P-game (§3.1), BPIP's advantage over AB seems to be increasing at higher search depths.

## 3.5 Warri

| | depth=1 | 2 | 3 | 4 | 5 | 6 | 7 | total 2-7 |
|---|---|---|---|---|---|---|---|---|
| NPU 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[12] if $\leq 16$ 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 NPU significantly at every search depth $\geq 2$, except for depth 6 where the result is not statistically significant[13].

Next, we ran tournaments at fixed search depths between BPIP 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.

| depth | AB wins | BPIP wins | draws | AB avg stones | BPIP 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 |

[11]BPIP used an older evaluation function not based on K-S trees.

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

[13] 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.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 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 BPIP is the superior statistical decision procedure.

### 3.6 Statistical dependencies potentially dangerous to BPIP-DFISA

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

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

and $\delta_2$ (defined similarly but for a node which is a sibling of the node that yields $\delta_1$) for $\approx 50000$ pairs of sibling nodes from BPIP search trees. We were using KS tree (§6.1.1) evaluators.

The observed centered correlation coefficients between $\delta_1$ and $\delta_2$

| | 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

BPIP'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 which 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, there will be important features left out of the evaluation function. This problem will, however, diminish the better the evaluation function is. Minimax and NPU suffer from related problems.

Notwithstanding the violation of independence, BPIP won every tournament at every depth[14] against both minimax and NPU, and thus appears to be the superior statistical decision procedure. BPIP's advantage

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

was sizeable in absolute terms in Othello, Warri, and Kalah, being almost worth an extra ply in the latter. NPU was able to beat minimax at some atypical games, such as the P-game and Kalah. However in games with long term features, such as Connect 4 and Warri (presumably also Othello) NPU is much worse than minimax, apparently because its neglect of correlations cause NPU to evaluate many nodes as near certain wins which are not.

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

In this subsection, we present experiments with more realistic gameplaying programs that utilize both the alpha beta (minimax) and BPIP-DFISA paradigms.

## 4.1 Othello

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

AB and BPIP are using the same evaluation function (AB is using the mean of the BPIP function, since AB requires a scalar) and AB is using response killer history tables and iterative deepening to do move ordering (similar to `BILL` [26]). The evaluator used a combination of linear regression and Kolmogorov-Smirnov 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 | BPIP Wins | Draws | AB Discs | BPIP 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 | BPIP evals/game | AB time used | BPIP 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.
- AB or BPIP wins - The number of games won by the player.
- Draws - The number of drawn games.
- AB or BPIP 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 $\sigma$ of confidence that BPIP is stronger than AB, based on win counts and (EQ 1) of §2.4. (A negative sign means AB is stronger.)
- Disc Conf. - The number of $\sigma$ of confidence that BPIP is stronger than AB, based on the number of discs won and (EQ 2) of §2.4. Usually more confidence is obtainable in this way.
- AB or BPIP evals/game - The mean and standard deviation of the number of positions evaluated by each player in a whole game.
- AB or BPIP 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, BPIP 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.

#### 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 BPIP search trees during a 200(300)-second tourney (rounded to integers). They may not represent any particular tree well. Thus in the 200-second tourney, the shallowest BPIP tree was only 1 ply deep and the deepest was 21 ply deep. (Remember that BPIP often chooses to get out of the search early if there is little utility in it.) Three *actual* BPIP 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 BPIP leaf was 6.86, but true line was searched by BPIP to an average depth of 6.94. This was still not as deep as the average depth reached by the AB player, 7.10, although possibly BPIP's search depth along the true game line was often deeper than AB's in situations where the move decision was difficult and important. In other situations, it is not clear that BPIP necessarily will search the true game line deeply. Leaves expanded by BPIP 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 BPIP thinks are of nearly equal value while all the rest are much worse, and BPIP 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 BPIP search trees only agreed with the true line for one move. That is, BPIP decided to make a move after expanding some of its siblings but not the move itself. BPIP won all of those games.

Note: average leaf depth figures for BPIP are difficult to interpret. For example, when BPIP decides to move in some easy situations with very little search, and save the time to search more deeply in other positions, it skews its average depth very much lower[15].

#### 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
BPIP used       Thinking time actually used by BPIP    (seconds/game)
```

The first line in each table is equal time consumption.

The more time BPIP 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 BPIP is given 50 seconds, AB needs about 1.14 'extra ply' to achieve parity, but if BPIP 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 BPIP 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 BPIP 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 BPIP player by about 1 disc per game, although it was still much stronger than the AB player[16]

#### 4.1.5 AB with "probcut"

Michael Buro[10] 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 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[17] outside the alpha-beta window. It then re-searches the moves which 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 [11] 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

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

[16] Even more statistical confidence – about $4\sigma$ – arose in earlier experiments with a weaker BPIP player that was more closely matched with its AB opponent.

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

| BPIP with 50 second time limit (giveaway factor ≈ 5) | | | | | |
|---|---|---|---|---|---|
| AB limit | AB used | AB ply | BPIP used | Conf. | **Disc Conf.** |
| 70 | 47.25 | 5.87 | 36.07 | 4.15 | 6.71 |
| 150 | 106.89 | 6.46 | 36.96 | 1.25 | 0.26 |
| 250 | 181.34 | 6.88 | 36.45 | 0.95 | 0.29 |
| 450 | 317.10 | 7.27 | 36.53 | -0.89 | -2.45 |

| BPIP with 100 second time limit (giveaway factor ≈ 14) | | | | | |
|---|---|---|---|---|---|
| AB limit | AB used | AB ply | BPIP used | Conf. | **Disc Conf.** |
| 120 | 84.58 | 6.26 | 81.35 | 5.10 | 8.24 |
| 200 | 144.24 | 6.69 | 82.42 | 3.98 | 6.25 |
| 400 | 282.99 | 7.22 | 83.11 | 2.61 | 4.43 |
| 800 | 552.43 | 7.67 | 82.43 | 1.48 | 1.95 |
| 1100 | 755.92 | 7.90 | 83.11 | 1.35 | 1.53 |
| 1600 | 1102.66 | 8.16 | 83.89 | 0.59 | 0.69 |
| 2100 | 1438.70 | 8.36 | 84.10 | 0.24 | -1.31 |

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

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

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

| S.T. logic | Time | AB Wins | BPIP Wins | Draws | AB Discs | BPIP 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 | BPIP evals/game | AB time used | BPIP time used |
|---|---|---|---|---|
| on | 193666 ($\pm$23099) | 124082 ($\pm$20356) | 84.58 ($\pm$7.13) | 81.35 ($\pm$9.18) |
| off | 161397 ($\pm$17079) | 112801 ($\pm$7675) | 70.89 ($\pm$10.03) | 71.27 ($\pm$3.81) |

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

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 was always significantly worse than BPIP for any value of $X$ that we tried.

| $X$ | ABP wins | BPIP 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 |
| $\infty$ | 23 | 109 | 10 | 27.09 | 5.10 | 8.24 | 6.26 |

Table 5: Othello results at 100 sec/side/game; AB+probcut vs BPIP. "Conf." is the number of $\sigma$ worth of confidence that BPIP 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.

| $X$ | ABP wins | BPIP 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 |
| $\infty$ | 19 | 118 | 5 | 25.41 | 5.87 | 12.20 | 6.81 |

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

#### 4.1.6 Results against independently written adversaries

We played matches between our BPIP 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

| $X$ | ABP wins | BPIP 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 |
| $\infty$ | 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` [26] 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$18) |
| | (4 draws) | (stddev 7.58) | (disc conf. 6.83) |
| program | wins | mean discs | sec/game consumed |
| BPIP | 131 | 41.88 | 226.99 ($\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.

| program | wins | mean discs | sec/game consumed |
|---|---|---|---|
| AB | 49 | 30.96 | 230.62 ($\pm$18) |
| Bugs | 89 | 33.04 | 293.53 ($\pm$4.5) |
| | (4 draws) | (stddev 7.16) | (disc conf. $-1.72$) |
| program | wins | mean discs | sec/game consumed |
| BPIP | 106 | 36.65 | 246.23($\pm$38) |
| Bugs | 27 | 27.35 | 292.97($\pm$5.4) |
| | (9 draws) | (stddev 5.86) | (disc conf. 9.46) |

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

Table 9 shows that `Bugs` is stronger than our AB program, but weaker than our BPIP 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[18]. BPIP with 5 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 ($\pm$16.38) |
| Bugs | 73 | 32.35 | 232.86 ($\pm$6.75) |
| | (9 draws) | (stddev 7.90) | (disc conf. $-0.52$) |
| program | wins | mean discs | sec consumed |
| BPIP(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 |
| BPIP(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. BPIP are in table 11.

| Time | Eclipse | BPIP | Draws | B/E sec | 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: BPIP vs. Eclipse. (3 games missing at 900 sec due to machine crash.)

BPIP is weaker than `Eclipse`, but not by very much. A plausible explanation why `Eclipse`'s performance relative to BPIP seemingly *improved* with more time from 100 $\rightarrow$ 1200 sec (while BPIP 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 BPIP 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 BPIP would gain ground on alpha-beta has merit, but the time controls need to increase a lot for BPIP 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, BPIP consumed less and less time in longer games – thinking on

[18] Although by a statistically insignificant margin.

average only 68% as long as Eclipse in the 20 minute games[19]. 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 BPIP 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 BPIP's 1350 evals/sec. At 240 sec/side/game, Logistello won 109 and BPIP 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. BPIP with 240) Logistello won 79 and BPIP won 55, with 8 draws (avg discs 32.69 to 31.25; time consumed 57 vs 186). Buro estimated BPIP would draw even with Logistello at 6:1 time odds. Considering the speed advantage of about a factor of 40 that Logistello has on our code, and that BPIP is using less of its time budget, at those 6:1 time odds BPIP 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 2-3 from its transposition table, which gain might also be realizable in BPIP. BPIP could be improved with a Logistello-class endgame solver. There are various other possibilities for speeding our BPIP search code[20]. BPIP'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.

## 4.2 Warri

Our latest BPIP Warri player is superior to our AB Warri player, see table 12.

| Time | AB Wins | BPIP Wins | Draws | AB Seeds | BPIP 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 | BPIP evals/game | AB time used | BPIP time used |
|---|---|---|---|---|
| 80 | 1236209 ($\pm$506316) | 291333 ($\pm$100913) | 75.34 ($\pm$29.18) | 73.49 ($\pm$25.35) |
| 160 | 2480482 ($\pm$990350) | 562644 ($\pm$199473) | 151.48 ($\pm$57.56) | 147.02 ($\pm$53.04) |
| 240 | 3758763 ($\pm$1549758) | 834577 ($\pm$347939) | 229.55 ($\pm$91.94) | 221.22 ($\pm$92.84) |

Table 12: Warri results at equal time usage.

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

[19]Where BPIP 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 BPIP was rarely in time pressure).

[20]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 21 of part 1. 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 BPIP gets 1300, a ratio of 19, not 40; 3. building a logistello-class BPIP player might require BPIP to reduce its high memory consumption, which might introduce further difficulties, 4. transposition tables may not cause as great a speedup for BPIP as for AB.

The advantage of BPIP 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[21]) based on tournaments we played but omit.

We report tree shape statistics:

```
AB tree statistics (from the 80 sec/game tournament above):
Search depth:            12.03    (+- 3.58)
Number of leaves:        17107.11 (+- 279543.35)

BPIP 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)
```

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 BPIP searches made by earlier BPIP 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. BPIP is using a 10% gulp fraction.

It was suggested to us that perhaps BPIP'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 $\geq 12$ seeds, and moves where the nonmover had no seeds on his side (both sides used the BPIP-mean evaluation function and 10-seed endgame tables), the quiescence player, suprisingly, 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 BPIP 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 BPIP Othello was evaluating about 1350 nodes/sec as compared with AB's 2300, we did not see a large increase in BPIP'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-BPIP Warri tournaments with an equal number of evaluations look rather good (see table 13) since they amount to cutting AB's time by a factor of 4.

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 BPIP was 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 BPIP Warri players both using as evaluation function for a leaf, the result of a BPIP search to depth $k$ on that leaf. (AB was given the mean of the distribution returned by the depth-$k$ BPIP search.) By altering the value of $k$, we measure the effects of artificially changing the "brainpower" of the evaluator.

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

| Evals | AB Wins | BPIP Wins | Draws | AB Seeds | BPIP 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 | BPIP evals/game | AB time used | BPIP 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 13: Warri results at equal evals/game.

This sort of 2-stage tree search is also of interest (see part I) for the purpose of reducing memory consumption in BPIP 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 BPIP (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 $2.34\sigma$ of confidence (based on seed count) that it was better than BPIP. At $k = 1$, BPIP won 32, lost 26, and drew 14, with $2.22\sigma$ of confidence (based on seed count) that it was better than AB. At $k = 2$, BPIP won 43-21-8, with $1.93\sigma$ of confidence[22].

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 $k$, BPIP's win counts monotonically improved, until with $k = 2$, when the evaluation function is about 17 times slower than at $k = 0$, BPIP won games at a 2:1 ratio.

This experiment supports our belief that BPIP gets stronger, relative to AB, when the evaluation function is slower and smarter. Slower evaluation function means BPIP 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 BPIP player here (with $k = 0$) was 2.9 times slower than AB, but lost, whereas our latest BPIP player, with smarter evaluator and better time control, is 4.0 times slower than AB, but wins.

## 4.4 Slagle Kalah

Even with our latest KS-tree (cf. §6.1.1,[52]) evaluation function for Slagle Kalah, BPIP 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 $\sigma$.

The AB player was searching to an average depth of 30 – enough to solve the game after not very many moves have gone by, and much deeper than our BPIP 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[52], 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 BPIP's, and all known evaluation functions for Slagle kalah are rather poor quality. All these conditions favor AB over BPIP.

BPIP 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 BPIP (using in fact an early, decision tree [but not our KS-tree], evaluator), BPIP still won 960-832-129, for $2.07\sigma$ of confidence. Since BPIP stores its search tree, one improvement in BPIP (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

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

BPIP for *new* node expansions. BPIP won a tournament reflecting such scoring by 1043-726-137, or $5.13\sigma$ of confidence, in spite of AB using 10% more evals.

## 4.5 Time control mechanisms

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

### 4.5.1 Othello

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

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

where $t_{\text{gulp}}$ is the estimated time to do the next gulp, $U_{\text{gulp}}$ 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 $\eta$ 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 $c_4 = 0.9$.

### 4.5.2 Warri

We found that with our current decision tree data, the BPIP Warri player believes that the utility of searching decreases as the game progresses. This may simply 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 BPIP 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 BPIP 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:

$T_{\text{tot}}$ Time left on BPIP's clock.

$M_{\text{tot}}$ Estimated number of moves left for BPIP to make.

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

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

$U_{\text{gulp}}$ Estimated utility of the next gulp.

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

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

$C_{\text{w}}$ 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 $C_{\mathrm{w}}(U_{\mathrm{gulp}} + U_{\min})/U_{\mathrm{final}} < (T_{\mathrm{used}} + T_{\mathrm{gulp}})M_{\mathrm{tot}}/(T_{\mathrm{tot}} - T_{\mathrm{gulp}})$, 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 $U_{\mathrm{final}}$ value in various ways. At present, we initialize $U_{\mathrm{final}}$ to a value somewhat lower than we expect it to be or the first move, based on a large number of games. After each BPIP move, it is updated to $U_{\mathrm{final}} = 0.9U_{\mathrm{final}} + 0.1U_{\mathrm{gulp}}$. Thus if the utility drops for several moves in a row, $U_{\mathrm{final}}$ will drop and the BPIP player will adjust its utility meter to spend more time thinking.

#### 4.5.3 Opponent-dependent time control and move decisions?

Chess master Gerald Seidler suggested that against a stronger or weaker opponent, it might be better to choose one's move to be the one with maximal $\mu + c\sigma$, where $\mu$ was the expectation value and $\sigma$ the absolute deviation of the BPIP probability distribution for each move, and $c$ is now a *nonzero* constant depending on the opponent. E.g. by choosing $c$ proportional to how much higher our rating is, we seek to "lose the fish in the complications." Against an equally strong, but nonidentical, opponent, "simplifying" with $c < 0$ might tend to move to positions we understand better than the opponent. Also, nonzero $c$ might be appropriate if one side is ahead on the clock.

As an experiment, we played Othello tournaments between our BPIP programs where one player was artificially made stronger or weaker by giving it twice as much (or half as much) thinking time, and where the other player made move choice decisions based on $\mu + c\sigma$ for various values of $c$. We also played the BPIP$(c)$ programs versus Slate's program with unequal thinking times.

The results [52] neither confirm nor disprove Seidler's theory, but do make it clear that any Othello benefit from Seidler's strategy is small. We have ignored this idea elsewhere in the paper.

## 5 Engineering tricks

Soon after the first alpha-beta chess players appeared, so did various engineering improvements upon them, including "quiescence" and [50] "iterative deepening." Although the rate has slowed, such improvements continue to appear even 40 years later [4, 10]. We similarly conjecture that there are many engineering tricks to be had in BPIP search.

### 5.1 The multispike trick

There is a tradeoff in BPIP 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. BPIP-DFISA 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 BPIP 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 BPIP. 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

BPIP contains a parameter called "gulp fraction" which specifies the fraction of leaves expanded each gulp. Table 14 shows the results of tournaments between our BPIP Othello player with various gulp fractions, and our alpha-beta player.

| Gulp size | AB sec used | BPIP 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 14: BPIP with various gulp fractions versus fixed AB player.

The best choice seems to be about 4%. The sensitive dependence on gulp size was confirmed by extensive experiments[23].

The following model yields insight into the sensitive dependence on gulp fraction. Say we use gulp fraction $f_1$, but the fraction of leaves which 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 footnote 12 of Part 1 might pay substantial dividends. The idea there was to achieve greater selectivity by ordering 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 BPIP to include partial node expansion may yield substantial payoffs.

This analysis also indicates that a still smaller gulp size might be preferable at longer time controls. Possibly this would further improve our results. The experiments reported in this paper were done using the 4% gulp size derived from tuning experiments at 100 seconds.

Notice also in table 14 that BPIP'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.

[23]That is, we had practically finished this paper running all tournaments at an arbitrarily chosen gulp size of 10% before investigating tuning. Switching to 4% improved our (already superior) time odds advantage against alpha-beta by another factor of about 4.

### 5.3 The large-utility trick

Occasionally single leaves occur with a sizeable 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.

### 5.4 Other tricks

As will be clear to those readers who examine our BPIP pseudocode... we often wrote space inefficient code, if by so doing we could keep it simple. We feel this was the right choice for a first implementation of BPIP, and it was good enough to beat alpha-beta. The long TR version [52] lists several improvements in memory management we know of, but mostly did not yet choose to implement.

## 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 one that simply returns a number approximating the expected game result if play were to continue from a given position. We have usually done this by multivariate linear regression. We devise a set of $n$ "positional features" and find weights so that the weighted sum of the features is a best (least squares) approximation to the final game result, over all positions arising in a very large database of games.

Let $X$ be an $N \times n$ matrix whose entries $x_{ij}$ represent the value of feature $j$ in position $i$ of a large database of positions. Let $y_i$ represent the final game result in the game in which position $i$ arose. We wish to find weights $\omega_{1..n}$ so that $y \approx \vec{x} \cdot \vec{\omega}$. The least squares choice of $\vec{\omega}$ is $\vec{\omega} = (X^T X)^{-1} X^T \vec{y}$. Of course [22], one wishes to find $X^T X$ and $X^T \vec{y}$ (total storage $O(n^2)$ numbers) without ever storing the much larger $N \times n$ matrix $X$.

Many-valued $y_i$'s gain more information and a better fit in games with multivalued final score. However, in games whose result is binary ($y_i \in \{0, 1\}$) Buro [10] has suggested use of "logistic regression" instead of linear regression.

The idea here is to devise an evaluation function of form

$$p = \frac{1}{1 + \exp(-\vec{x} \cdot \vec{\omega})} \tag{6}$$

(where $\vec{x}$ is the vector of features of some position, and $\vec{\omega}$ is a vector of fitted weights) estimating the *probability of winning* a position with those feature values. The maximumum likelihood statistical religion then proposes choosing $\vec{\omega}$ to maximize $\prod_{\text{wins}} p_i \prod_{\text{losses}} (1 - p_i)$. [54] gives an iterative reweighted least squares process which converges to this $\vec{\omega}$. We will now derive and describe a (presumably) equivalent algorithm.

We seek to maximize the log likelihood $L$ by choice of $\vec{\omega}$:

$$L = \sum_i \ln \frac{1}{1 + \exp(\mp \vec{x}_i \cdot \vec{\omega})} \tag{7}$$

where the upper signs are taken if position $i$ is a win and the lower ones if it is a loss. By taking second derivatives, one may verify that $L(\vec{\omega})$ is a concave-down function, hence has a unique maximum. By taking first derivatives, we see that the optimal $\vec{\omega}$ satisfies

$$\frac{\partial L}{\partial \vec{\omega}} = \sum_i \frac{\pm \vec{x}_i}{1 + \exp(\pm \vec{x}_i \cdot \vec{\omega})} = \vec{0}. \tag{8}$$

Thus (since we have an expression for the gradient) we can do steepest ascent to approach the optimum. If we linearize this gradient with respect to $\vec{\omega}$ in a neighborhood of $\vec{\omega}_0$, we have that

$$\frac{\partial L}{\partial \vec{\omega}} = \sum_i \pm \left[ \frac{1}{1+\exp(\pm \vec{x}_i \cdot \vec{\omega}_0}) - \frac{\vec{x}_i \cdot (\vec{\omega} - \vec{\omega}_0)}{2 \ + \ \exp(+\vec{x}_i \cdot \vec{\omega}_0) \ + \ \exp(-\vec{x}_i \cdot \vec{\omega}_0})\right] \vec{x}_i \ + \ O(|\vec{\omega} - \vec{\omega}_0|^2) \tag{9}$$

An ultimately[24] quadratically convergent iterative process is thus to start with $\vec{\omega} = \vec{0}$ and then produce the new $\vec{\omega}$ each iteration by solving the *linear* system

$$\sum_i \pm \left[ \frac{1}{1+\exp(\pm \vec{x}_i \cdot \vec{\omega}^{\text{old}})} - \frac{\vec{x}_i \cdot (\vec{\omega} - \vec{\omega}^{\text{old}})}{2 \ + \ \exp(+\vec{x}_i \cdot \vec{\omega}^{\text{old}}) \ + \ \exp(-\vec{x}_i \cdot \vec{\omega}^{\text{old}})} \right] \vec{x}_i = \vec{0} \tag{10}$$

for $\vec{\omega}$, where $\vec{\omega}^{\text{old}}$ is the old value $\vec{\omega}$ from the previous iteration. The value of $\vec{\omega}$ calculated during the first iteration of this process is proportional to the usual least squares linear fit weights, but subsequent iterations alter that.

Buro tried the logistic procedure ("LOG") and the usual linear least squares fit ("FISHER") to devise an Othello evaluation function based on a large database of positions and win/loss data[25]. He then played a color-balanced tournament between the two resulting (minimax based) Othello programs, and LOG won, 40-30. (LOG also garnered 43.5 points versus QUAD, a third program, while FISHER only got 41.5 points against QUAD.) Buro later got further small improvements by imposing certain constraints of the form $\omega_i \geq 0$ (or $\leq 0$) on some of the weights and finding the maximum likelihood estimator subject to these constraints. Unfortunately these Buro results are only statistically significant at the 10% level (cf. §2.4), but in the absence of other data this may still be enough to convince many game programmers to prefer logistic regression.

In fact we have often used *piecewise* linear regression; we use several vectors of weights, the relevant one depending on the "game stage." (Similarly Buro used piecewise logistic regression.) In some cases smoothing of the boundaries between game stages was used. 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.

For games involving "quiescence," one should only fit quiescent postions, since the search only evaluates such and unquiescent positions add noise. We attempted to deduce the values of the pieces in chess from 175000 master games. Fitting all positions to game results, we found:

$$P = 2.3, \ \ N = 2.6, \ \ B = 3.4, \ \ R = 4.7, \ \ Q = 8.1. \tag{11}$$

When "unquiescent" positions (those in which the next move was a capture, check, en passant, or promotion) were removed from the dataset, our fitter found

$$P = 1.30, \ \ N = 2.82, \ \ B = 3.18, \ \ R = 4.54, \ \ Q = 8.80. \tag{12}$$

which is much closer to the usual chessbook values

$$P = 1, \ \ N = 3, \ \ B = 3, \ \ R = 5, \ \ Q = 9. \tag{13}$$

In fact, our fit is *equivalent* to chessbook values in that we value every unequal exchange of a small set of pieces with the same sign (e.g. $P + B < R$, $2P < N$), *except* that our fit will sacrifice 2 N's to get 5 P's. Our fit[26] may be more accurate than the book view here.

[24]Convergence is *not* assured, if the starting point is poor enough. But one can easily modify the iteration to, for example, always go to the max-quality point on the line segment with endpoints $\vec{\omega}$ and $\vec{\omega}^{\text{old}}$, and that algorithm will always converge, since it continually ascends.

[25]It is unclear whether Buro's decision to ignore final scores was wise, but his experiment is still revealing. Actually Buro used Fisher's 2-class linear discriminant, which is not quite the same as the least squares. They are the same up to a constant factor – but the constant may change with game stage, hence the resulting *piecewise* evaluators could be inequivalent.

[26]We tried 8 other fitting methods, e.g. fitting to evaluations of successor positions instead of to final game result, but the fit of quiescent positions to final game result gave the best result.

We only accepted features which passed various tests of statistical significance. First, each feature's weight had to have absolute value at least 10 times its standard error. (Our least squares fitter, which is based on a procedure by Jennrich [22], calculates these standard errors and automatically refuses to incorporate features into the fit if they do not pass such "F-tests.") Features which often were thrown out of fits were suspect. Third, features with small "importance" (weight times standard deviation) were suspect. Fourth any features that come in matched pairs but behave unlike their compliment aroused suspicion. Fifth, 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.

This procedure can be quite painful but it was the one we used to make our initial checkers, mod-9-connect-4, and Othello scalar evaluators.

To get relative accuracy of $\epsilon$ when fitting $n$ weights using $N$ datapoints, simple probabilistic analysis requires $N \gtrsim cn/\epsilon^2$ where $c$ is a constant depending on the confidence one desires of obtaining the desired accurary, the probabilistic properties of the data, and the importance of the various features in the final fit. In the best case, the features are independent unbiased coin flips of equal importance, in which case $c$ will be of order 1. (To see this, consider estimating one of the weights just from its feature, and ignoring all the other features.) But if the coin flips are biased an amount $0 < B < 1/2$, have importances ranging from $\omega_{\min}$ to $\omega_{\max}$, and are correlated so that their covariance matrix has condition number $\kappa$, then $c \approx \kappa\omega_{\text{avg}}/(B\omega_{\min})$. Even if the correlations are so severe, that, e.g., all coin flips are the same except that one out of the $n$, selected at random, is random (and say all the weights are equal) then $\kappa \approx n$, so the growth of $\kappa$ with $n$ can be expected to be at worst polynomial rather than exponential. Thus Acton's complaints ([1] page 253-4) about people who want to do 300-parameter least squares fits do not seem to be justified in our case.

Roughly speaking, our actual experience is that, with care, 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.

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.

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 TM [52].

### 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 BPIP 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 between these two values, $\Delta$, is the "opinion change."

We then wish to develop, by learning from this dataset, an evaluator which will return the probability distribution of $\Delta$ 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 $\Delta$. 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," [51, 29] applied to *uniquified*[27] 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 a $\approx 10^5$-point dataset in $\approx 30$ 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 (§6.1.5), might be better.

### 6.1.2 Compression of probability distributions

The distributions returned by our Kolmogorov-Smirnov tree evaluator typically contain a large number ($\approx 200$) of spikes. For BPIP 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 [18].

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 to large for numerical stability.

Secondly, methods which preserve moments can exhibit problems when the data contains rare outliers as 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 which 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 BPIP, 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 have had positive experiences with two different "learn as you play" methods for constructing evaluators.

The first method is simply to pick positions from BPIP search trees (as one plays games) at random, with some fixed low probability. For each such position, the opinion change $\Delta$ is computed. Assuming we already have a KS-tree based (or other bin-based) distribution-valued evaluator, this evaluator may then

[27] 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.

be improved by inserting this $\Delta$ value inside the dataset in the appropriate bin. This improvement process can be run forever and will gradually improve the accuracy of the probability distributions returned by the evaluator[28]. Since we use a moment based compressor (c.f. §6.1.2) we can keep the space requirements small by only updating the *moments* of the relevant distributions. Also, by making the probability small enough, this learning procedure entails virtually no runtime overhead.

We did an experiment pitting BPIP with utility-guided tree growth and an early learning decision tree evaluator ('B') versus an early, and nonlearning, minimaxing opponent ('A') in Slagle Kalah. Performance gradually improved over the course of 5 tourneys (tourney 0 involved no learning; the other tourneys involved the evaluator being updated after each game):

| 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 |

Note the clear trend of improvement over the course of only 310 games (8-9 seconds total B-thinking time per game).

The second learning method may be used to improve the weights in linear regression based scalar evaluators, or to learn table-based scalar evaluators, or with nonlinear evaluators. (The table-based evaluators we are considering are simply the sum of some table lookups, and thus may be regarded as a linear evaluator in a very high-dimensional space.)

At each interior node in a search tree, one evaluates the node both by means of the scalar evaluator, and also by one's normal lookahead procedure. (In alpha-beta search, one sometimes only gets a lower or upper bound on the lookahead value, not the value itself.) One then makes a small adjustment to the evaluator weights (or table entries) designed to decrease any disagreement between these two.

In our table based evaluators, the update rule we used was simply to add an adjustment to the relevant table entries, the value of this adjustment being chosen to reduce the disagreement by some small constant fraction $f$. (In Othello, we used $f = 1/10$. In Warri, we used $f = 1/64$. Under reasonable assumptions it may be shown that the RMS size of the noise in a table entry, assuming that the noise in its lookahead [training] values was of order $\eta$, tends to order $\eta\sqrt{f}$.) More generally one could move a fixed fraction of the steepest descent direction (in an appropriately scaled space) or Newton direction (§6.1.5). Empirically one finds that, if the constant is made small enough, convergent behavior is obtained.

Because this update is efficient and because it only is applied at *interior* nodes of the search tree, a game player with learning will have only a small runtime overhead compared to a player without, so you never lose much by putting in this sort of learning, but the potential gains can be significant. The point of this method is that tremendous update rates can be obtained (comparable to the node rate in the searcher), allowing much more learning to occur than one could ever get learning from any fixed database.

We tried this update method in Othello and Warri. For nonterminal positions in Othello, we used an evaluator which simply added up a set of tabulated values, one table entry for every possible state of every possible line (there are $3^k$ states per $k$-long line, $k \leq 8$; we considered "bounce diagonals" to be "lines") on the Othello board, and with different sets of tables at 6 different game stages. (The point being that many features considered to be important in Othello, such as mobility, number of frontier squares, and edge states, could be gotten from tables of this sort, either exactly or approximately.) Our evaluator tables in total constituted 708588 bytes. Because this evaluator is so simple, we can obtain rates of 70000 evaluations/second during searches – enormous rates. Starting from a set of tables with all entries *zero*, we conducted learning negascout [35] searches with transposition table. After a few hours of learning, it was apparent that the program had learned *something* (it would make feeble efforts to avoid giving up a corner)

[28]The point is that the data is being sampled from the right probability space, although the definition of this space is continuously changing! One could also consider learning methods which dynamically alter the tree topology in response to new data, as opposed to merely altering its contents.

but it 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, searching about 12 ply, is somewhat weaker than Internet Othello Server programs with strong evaluators which search only 6 ply. These in turn are weaker than the other AB and BPIP Othello programs we wrote which combined handmade features via a linear regression. Therefore, although `obaby`'s learning feats were impressive, its learned table-based evaluator was still weak in an absolute sense. Therefore, we decided to abort `obaby` and stick with an evaluator based on handmade features and linear regressions.

In Warri, we had more success with the table learning technique (see [52]) than in Othello, and hence we used it in our final Warri program. This success, as compared with Othello, was probably due to the following differences between Warri and Othello.

1. we started Warri learning from a fairly good evaluator, instead of to all zeroes, and the search can access a large source of perfect Warri knowledge (endgame tables), which has no analog in Othello.
2. the node rate in our Warri learner is over twice as fast as in Othello, and the effective branching factor over twice as small – leading to faster learning,
3. the tables were over 10 times smaller than in Othello – less to learn,
4. since there is a comparative dearth of human understanding of Warri, it is much harder to design good Warri features to put in a competing nonlearned evaluator.

#### 6.1.4 The importance of being earnest

At first, we naively believed it would suffice to train our scalar evaluators on positions arising in AB searches, or positions arising from performing 1 or 2 random moves in positions from games, or even just on positions from games. Recall the evaluator is supposed to predict expected game result for positions arising in BPIP searches. It turns out to be important to sample positions actually arising in such searches. The alternatives above all work substantially worse[29].

We also solved Slagle Kalah (§7.3), and so tried training a scalar evaluator on a large set of positions randomly sampled from the proof tree of Slagle Kalah, fitting to *game theoretic* values. As predicted by the BPIP philosophy, this produced poorer results than training from actual game results in a large set of imperfectly-played games.

#### 6.1.5 Other possible learning methods

The methods we have described (linear regression, Kolmogorov-Smirnov decision trees) for self-learned evaluation functions were adequate for our purposes, but may not be the best possible methods. We now discuss three other possibilities, in order of increasing speculative character.

First, **entropy trees**[30] may be preferable to Kolmogorov-Smirnov decision trees. One wishes to divide the data into subsets by means of questions arranged in a decision tree. At each node choose choose the question minimizing

$$p_1 S_1 + p_2 S_2, \tag{14}$$

where $p_1$ is the probability that the answer to the question (for a random element of the dataset) is "yes" (and $p_1 + p_2 = 1$) and $S_1$ is the information theoretic entropy of the corresponding subset of the dataset,

[29] And were listed in decreasing order of quality. Recall the learning experiments from §6.1.3. The 'B' player in tourney 0 had been trained on positions from AB searches.

[30] This idea is well known [17, 34], but we believe our growth termination condition is novel.

regarded as a probability distribution. A convenient numerical estimate of the entropy of a (possibly mixed continuous and discrete) probability distribution, from which we have $N$ samples in sorted order along the real line (let the number of occurrences of a sample at $x_i$, be $m_i$, for $i = 1..n$, so that $\sum_{i=1}^{n} m_i = N$) is

$$S \approx \log_2 N - \sum_{i=1}^{n-1} \frac{m_i + m_{i+1}}{2N} \log_2 \frac{m_i + m_{i+1}}{2(x_{i+1} - x_i)}. \tag{15}$$

Apply this procedure recursively until left with a complete decision tree whose leaves are singleton datasets. One then "prunes back" the decision tree by repeatedly combining any two leaves for which

$$B < N|\Delta S| \tag{16}$$

where $B$ is the number of bits needed to describe the question which distinguishes the two leaves (e.g. if there were $Q$ possible questions, perhaps $B = 2 + \log_2 Q$ is a sensible value to use), $N$ is the number of data samples we have at that tree node, and $\Delta S = S_{\text{parent}} - p_1 S_1 - p_2 S_2$ is the entropy reduction due to that tree node. This termination condition is the one which naturally arises from the large-$N$ asymptotics of E.T. Jayne's "entropy concentration theorem" [21] combined with the notion that the prior among models of description length $B$ is proportional to $2^{-B}$. The philosophy is that an increase in model complexity of $B$ bits is justified by improving the description of $N$ samples by entropy change $\Delta S$ bits, where (EQ 16) holds.

Appealing features of entropy trees, as opposed to KS trees, include:

1. They attempt (albeit greedily) to maximize a global tree quality measure: the reduction in the information theoretic entropy of the data, induced by the tree, minus the description length of the tree. This is like the "minimum description length" religion except using the entropy, rather than the description length, of the data;

2. Entropy trees may be computed quickly by an "incremental" approach. One needs a subroutine to compute the change in $\Delta S$ arising from changing the question. If all questions are of the form "is feature $i$ greater than $X$?" and we are scanning $X$, elementary data structuring tricks allow such a subroutine to run in $O(\log N)$ time. As a result, the greedy question may be deduced, in an $N$-sample $d$-dimensional dataset, in $O(Nd \log N)$ steps, and the entire tree may be built (assuming that the questions always produce roughly equal dataset splits) in $O(Nd \log^2 N)$ steps.

As an experiment, we programmed entropy trees in Othello, and they seemed to produce probability distribution valued evaluation functions of comparable or better quality than the ones produced by KS trees. Entropy trees may have many applications outside of computer game playing.

Second, a well-known method called "stochastic ascent" can be used to do **nonlinear regression**. To optimize a nonlinear quality criterion, we use the fact that the gradient of the quality measure on a *single, randomly selected* game position extracted from the dataset, is an unbiased estimator of the desired dataset-wide gradient. At the $n$th iteration, one walks the vector of fitting parameters a distance proportional to $1/n$ in this local gradient direction. This update learning can be embedded inside the tree search as in §6.1.3.

Third and finally, we mention an idea inspired by hearing a talk by R. Sutton: **evaluation function learning as a Markov model.** One views the integer $n$-vector of positional features seen by the evaluator as dividing the set of possible game positions into subsets; each subset is associated with a particular point of the $n$-dimensional grid. After a single move, the positions in one grid point could change into ones associated with other grid points. With each grid point $x$ we associate an expectation value $E_x$ (of game result, given that we are in a position in the class $x$ sampled from a typical game position) and transition probabilities $p_{xy}$ that the next move will cause one to enter a position of class $y$.

The problem of learning a *scalar* evaluation function is then to deduce the $E_x$, and the problem of learning a *probability density valued* evaluation function suitable for BPIP-DFISA is also to deduce the $p_{xy}$. These quantities obey the relations $E_x = \sum_y p_{xy} E_y$, which, for example, would permit the $E_u$ to be deduced from the $p_{xy}$ and the game-end positions. (As a matrix and vector equation, this is $P\vec{E} = \vec{E}$ and $\vec{E} = P^{-\infty}\vec{G}$

where $\vec{G}$ is the vector of game end values. In other words $\vec{E}$ is the eigenvector of $P$ with [least modulus] eigenvalue 1.)

Unfortunately, the number of possible grid points is generally not going to be small enough to actually store estimates of all the $E_x$, much less all the $p_{xy}$. If it were this small, then it would be an easy matter to estimate these quantities (by counting). Instead, we must *model* the $p_{xy}$ and $E_x$.

# 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 [52].

## 7.1 Othello

For the rules of Othello, see [24] or [38]. An important rule not mentioned by these sources is the scoring of games which 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 [39] and [16]. 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 [39] and from [12].

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. [48, 49], who used it as a vehicle for studying game tree search, and studied by other AI researchers (see eg [14]). See [48, 49] 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[31].

### 7.4 Mod-9 connect 4

Mod-9-Connect-4 is played on a $9 \times 9$ 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 unnoccupied 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 $6 \times 7$ 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, 53]. 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, BPIP seemed able to beat alpha-beta on an even playing field, or even giving sizeable time odds, with increasing advantage as the game became more complex or the time controls became longer.

We would like to propose, tentatively, as an empirical law holding for many typical games, that BPIP with utility guided growth thinking for time $t$ will 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 $P > 0$. This would correspond to the prediction that BPIP search with utility-guided tree growth is asymptotically about as powerful as plain AB search going a *constant factor deeper* than it normally would in time $t$. It would also correspond to the prediction that the loglog plot of figure 1 asymptotically looks like a straight line[32].

We have a simplistic model which explains this law. Model the children of a tree node as having 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 amount. This idea was basically thought of a long time ago by R.W. Floyd, who proposed making and using an a priori heuristic plausibility function for this very purpose. If $c$, $0 < c < 1$, is any constant, then the $N^c$ deepest (among the $N$ total) Floyd lines will penetrate a constant factor deeper than average, with probability$\rightarrow 1$. Presumably one cannot afford to misestimate this many lines without risking making the wrong move – leading to the desired law.

From the point of view of the programmer, BPIP and alpha-beta programs have some differences. A crude program based on BPIP-DFISA 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 BPIP program. In an alpha-beta program, you need to work on the evaluator, search heuristics, and time control heuristics. Top alphabeta

[31] 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.

[32] In our Othello experiments [cf. table 3] the slope of the line in figure 1 is $P \approx 1.5$, and thus AB would need to go to depth $\approx 2.5d - 7.7$ instead of its normal depth $d$ to equal BPIP. This is all tentative since it based on only 4 datapoints, the last of which is more an extrapolation than a measurement.

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. BPIP takes care of shaping the search and dealing with time control issues, so you don't have to. In BPIP you can focus on developing the evaluator.

Obvious things to try next might include:

- Transposition tables – how should they best be implemented in BPIP, how should one best handle the issues related to BPIP in DAGs (cf. §11 of part I), and how does all this affect performance?
- 2-stage BPIP search to reduce memory consumption (§12 of part I).
- Distribution compression within the BPIP search to reduce time and space needs at the sacrifice of some accuracy.
- Variants of BPIP with partial node expansion.
- Deeper, guided expansion of very high utility nodes.
- Further investigation of automated statistical methods for generating evaluation functions for BPIP.

And then, one might want to try writing a BPIP chess program.

In yet another direction, we wonder if BPIP 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] Leo Breiman, J.H. Friedman, R.A.Olshen, C.J. Stone: Classification and regression trees, Wadsworth 1984

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

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

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

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

[14] 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.

[15] 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.

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

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

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

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

[20] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

[34] 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.

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

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

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

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

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

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

[41] S. Russell, personal communication.

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

[43] 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.

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

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

[46] 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.)

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

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

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

[50] 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

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

[52] 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.

[53] 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

[54] 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.

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

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

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

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

[59] Brian W. Kernighan, Dennis M. Ritchie: The C programming language, second edition, Prentice-Hall, Englewood Cliffs NJ 1988

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

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

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

[63] Bjarne Stroustrup: The C++ programming language, Addison-Wesley, Reading MA 1991

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

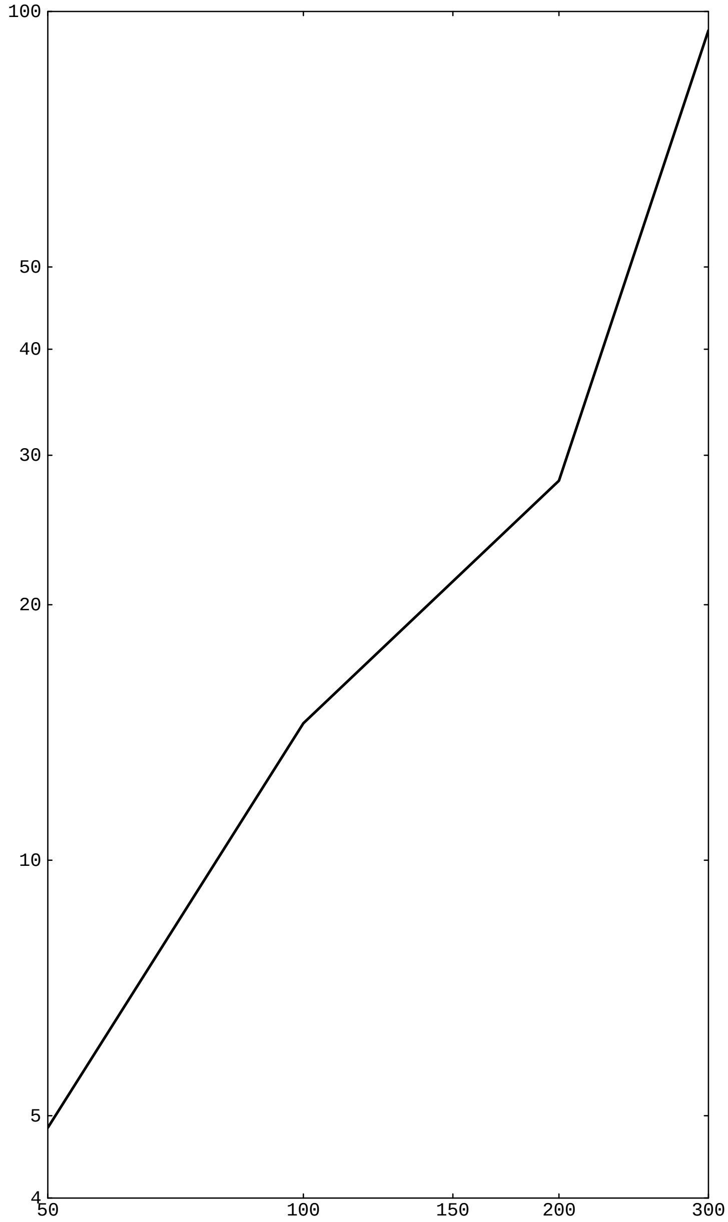


Figure 1: Loglog plot of allowable BPIP-AB time consumption giveaway factor (vertical axis: our best guess, based on time odds tourney table ; readers may conjure up their own error bars...) in Othello, versus BPIP thinking time per game (seconds; BPIP evaluated 1350 nodes/sec as compared with AB's 2300).

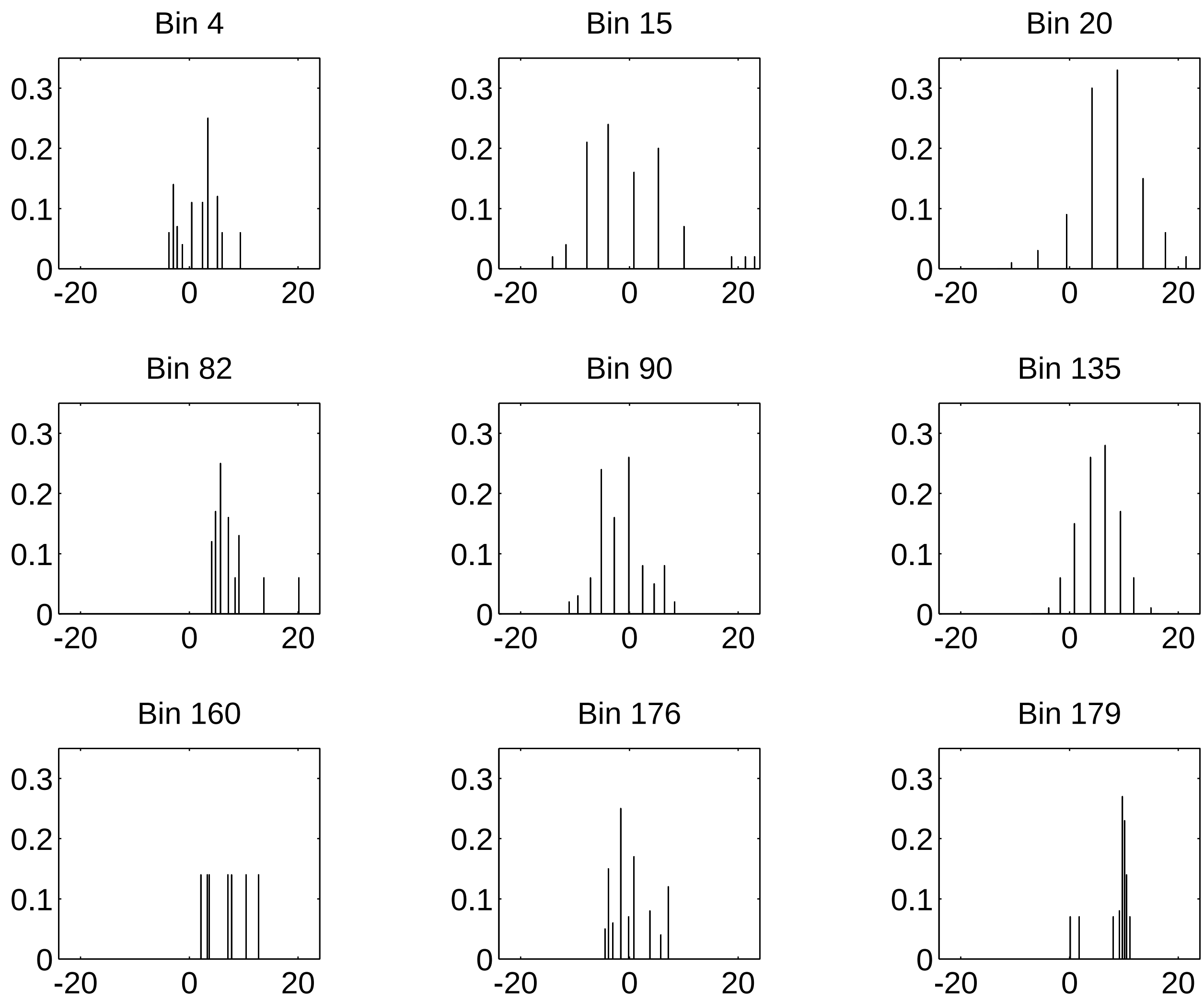


Figure 2: Pictures of 9 distributions from our Othello player's KS tree bins, produced by moment based compression of opinion change data at depth 5-6.