ProbCut:
An Effective Selective Extension of the $\alpha\beta$ Algorithm

Michael Buro[1]

Paderborn, Germany

Abstract

This article presents a new, game-independent selective extension of the $\alpha\beta$ algorithm. Based on the strong correlation of evaluations obtained from searches at different depths it is shown how the result of a shallow search can be used to decide with a prescribed likelihood whether a deep search would yield a value outside the current search window. In its application to Othello, the technique is shown to be effective in investigating the relevant variations more deeply. It significantly increased the playing strength of an already strong brute–force Othello program.

1. Introduction

Human players tell us that a game tree does not have to be searched in its full width in order to find a good move. Using their experience, they are able to prune in advance unpromising variations. The resulting game trees are narrow and can be rather deep. By contrast, the original minimax algorithm searches the entire game tree up to a certain depth and even its efficient improvement — the $\alpha\beta$ algorithm (Knuth & Moore (1975)) — is only allowed to prune backwards since it has to compute the correct minimax value.

In what follows, a selective extension of the $\alpha\beta$ algorithm — ProbCut — will be presented which gives the common brute–force approach a human touch. However, it is not the first step in this direction. Besides selective quiescence–search methods, such as the null–move heuristic (Beal (1990)), other algorithms have been proposed — for example by McAllester (1988), Palay (1985), and Rivest (1988) — which are able to search the game tree in a best–first manner, but use an amount of memory roughly on the order of number of nodes searched. This is not practical for programs with a fast evaluation function running on conventional hardware. ProbCut, however, needs no memory and its application is not limited to quiescence search. Also, it is effective not only in tactical positions where one move is clearly superior to all others, such as the "singular extensions" introduced by Anantharaman et al. (1990).

A technique with aims similar to ProbCut is used in the checkers program CHINOOK. Schaeffer et al. (1992) described their approach, casually, in a footnote: CHINOOK performs forward cuts in positions with a material deficit in which a shallow search does not show an escape. ProbCut is a generalization of this method in that it is game independent and does not rely on parameters to be chosen by intuition. It was tested by incorporating it in the author's already strong Othello program LOGISTELLO[2] and increased the program's playing strength considerably.

[1]University of Paderborn, Department of Mathematics and Computer Science, 33095 Paderborn, Germany, Email: buro@uni–paderborn.de

[2]A description of LOGISTELLO is given by Buro (1994).

2. Probabilistic forward cuts

The selective extension presented here permits excluding probably irrelevant subtrees from being searched deeply. The time so saved is used to analyze the relevant variations more deeply, given some constant available search time. Figure 1 shows an outline of the negamax $\alpha\beta$ algorithm computing a position's value from the point of view of the side to move. For clarity, a global variable `pos` is assumed to contain all positional information. The modification is based on the following idea: in order to evaluate a position using a deep search of depth d, the position can first be examined by a shallow search of depth $d' < d$. The result v' can then be used to estimate the true value v and to decide with a prescribed likelihood whether v lies outside the current `(alpha,beta)` window. If so, the position is not searched more deeply and the right–hand bound is returned. Otherwise, the search is performed to depth d, yielding the true value. Here, a shallow search has been invested but relative to the deep search the effort involved is negligible.

A natural way to estimate v by means of v' is to use a linear model of the form $v = a \cdot v' + b + e$ with $a, b \in \mathbb{R}$ and a normally–distributed error variable e with mean 0 and variance σ^2. If the evaluation function is relatively stable, it can be expected that $a \approx 1, b \approx 0$, and σ^2 is small, that is the accuracy of the unbiased estimator $\hat{v} = a \cdot v' + b$

```
int AlphaBeta(int height, int alpha, int beta)
{
  int i, max, val;
  POSDELTA delta;

  if (Leaf(&pos, height))            /*     leaf-position? */
    return Eval(&pos);               /* yes => evaluate it */

/* location of the selective extension */

  max = alpha;                       /* initialize maximum */
  for (i=0; i < pos.movenum; i++) {  /*   forall moves ... */
    Move(&pos, pos.move[i], &delta); /* make move and save */
                                     /*   changes in delta */
    val = -AlphaBeta(height-1, -beta, -max);    /* negamax */
    Undo(&pos, &delta);                 /* restore old pos */
    if (val > max) {
      if (val >= beta) return val;          /*      cutoff */
      max = val;                            /* new maximum */
    }
  }
  return max;
}
```

Figure 1: A negamax implementation of the $\alpha\beta$ algorithm

of v is high. By using the following equivalences it can be tested whether $v \geq \beta$ holds with a given probability:

$$v \geq \beta \iff \hat{v} + e \geq \beta \iff (\hat{v} - \beta)/\sigma \geq -e/\sigma.$$

Since $-e/\sigma$ is normally–distributed with mean 0 and variance 1 (and distribution function Φ), it follows that $v \geq \beta$ holds with probability of at least p if and only if $(\hat{v} - \beta)/\sigma \geq \Phi^{-1}(p)$. This condition is equivalent to $v' \geq (\Phi^{-1}(p) \cdot \sigma + \beta - b)/a$. Analogously, it can be shown that $v \leq \alpha$ holds with probability of at least p if and only if $v' \leq (-\Phi^{-1}(p) \cdot \sigma + \alpha - b)/a$. Thus, the only matter of interest is whether v' is greater or less than a value which depends on α, β, and the constants a, b, σ, and p by a simple relation. This observation immediately leads to the implementation of the ProbCut extension shown in Figure 2. Nodes at height d (=D) are first evaluated by means of a zero–window search of depth d' (=DP) to decide whether $v \leq$ `alpha` or $v \geq$ `beta` with probability of at least p. In this case, the corresponding bound is returned.

3. Determination of the parameters

If the game tree is searched with depth $d'' \geq d$, then the brute–force depth achieved by the new algorithm is $d'' - (d - d')$. On the other hand, the maximum depth is not greater than $d'' + (d - d')$ if the same time is used which the plain $\alpha\beta$ algorithm needs for a depth d'' search, since here depth d'' is searched completely, too. Choosing a large value for `PERCENTILE` leads to the original search behaviour up to a small time overhead due to the

```
#define PERCENTILE 1.5      /* i.e. p ca. 0.93            */
#define DP         4        /* depth of shallow search    */
#define D          8        /* check height               */

int AlphaBeta(int height, int alpha, int beta)
{
  ...
  if (height == D) {
    int bound;

/* v >= beta with prob. of at least p? yes => cutoff */

    bound = round((+PERCENTILE * sigma + beta - b) / a);
    if (AlphaBeta(DP, bound-1, bound) >= bound) return beta;

/* v <= alpha with prob. of at least p? yes => cutoff */

    bound = round((-PERCENTILE * sigma + alpha - b) / a);
    if (AlphaBeta(DP, bound, bound+1) <= bound) return alpha;
  }
  ...
}
```

Figure 2: The ProbCut extension

shallow searches, because there are only very few forward cuts. In the other extreme case, there are many (erroneous) cuts — the depth–d'' search then degenerates into a search with depth $d'' - (d - d')$. Hence, a playing strength similar to that of the non–selective program can be expected in this case, too, provided iterative deepening is used.

Before the parameters a, b and σ can be estimated by a linear regression, the search depths d and d' must be chosen. If the difference $d - d'$ is large, the variance of the error variable is large and this reduces the number of cuts. On the other hand, the difference must not be too small, because it is a measure of the savings that are achievable. Furthermore, while choosing d one has to take into account that this depth will be reached in tournament games and a sufficiently large number of evaluation pairs can be determined in a reasonable time. Experiments finally led to the choices $d' = 4$ and $d = 8$ for LOGISTELLO. It turned out that the variance of the error variable increases with the number of discs on the board, which is a natural measure of time in Othello. Therefore, the parameters a, b and σ were estimated for each game phase. In order to do this, evaluation pairs (v', v) were determined from positions which occurred exactly at the depth at which the test with the shallow search take place. In preceding tests it turned out that, under tournament conditions, the selective search normally reaches depth 13 or 14 in the opening and midgame phases on a Sun SPARC–10/M30 workstation in contrast to depth 11 or 12 formerly achieved by the brute–force program. Thus, positions reached from tournament positions at depth 6 by a normal $\alpha\beta$ search were used to estimate the parameters. Figures 3 and 4 show ca. 2,000 evaluation pairs[3] with the corresponding parameters and linear approximation at 28 and 44 discs, respectively. The goodness of fit is not only visually obvious but also the regression's coefficient of determination — $\hat{r}^2$ — of about 0.97 indicates that only some 3% of the variance in the data is ascribable to the random error. Therefore, the linear model is clearly suitable.

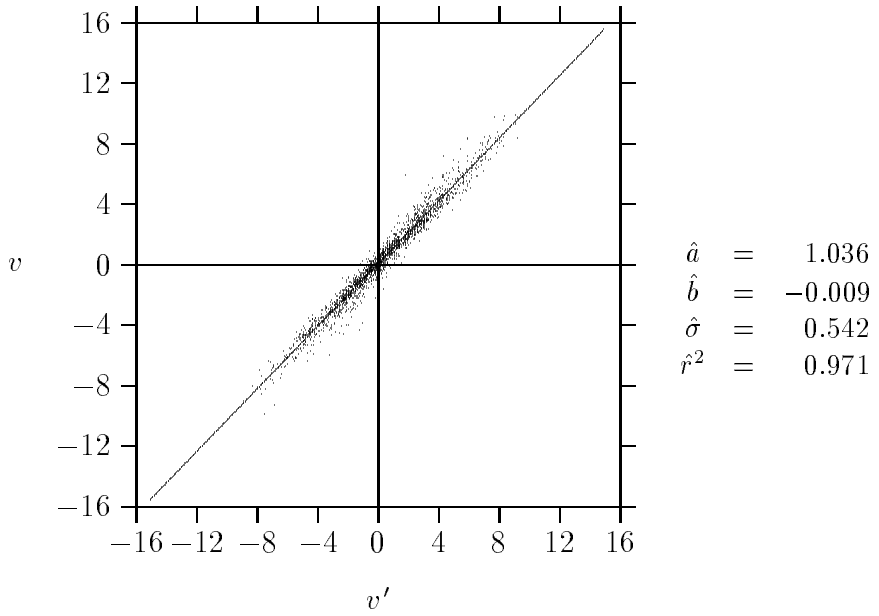

Figure 3: Relation between v and v' at 28 discs

[3]LOGISTELLO's evaluation function approximates the log odds of winning — $\log_e(P/(1-P))$ — with a resolution of 0.0001. For instance, value 2 stands for winning probability $P = 0.88$.

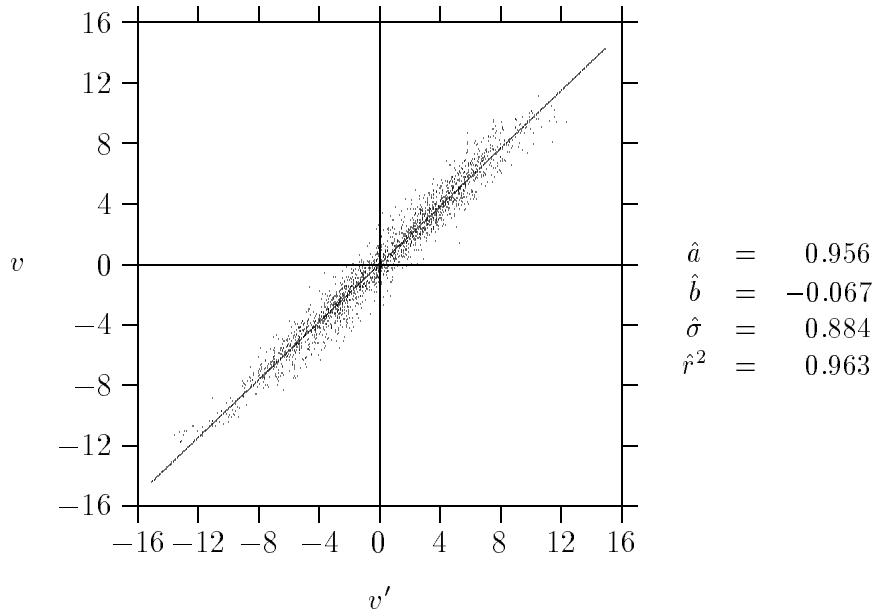

Figure 4: Relation between v and v' at 44 discs

It remains to choose PERCENTILE suitably. For this purpose, some tournaments between the non–selective and versions of the selective program were played, where it is understood that *all* program versions are selective in their quiescence search. Starting with 35 balanced opening positions[4] with twelve discs, each game and its return game was played under normal tournament conditions — 30 minutes per player per game — with a view of selecting the value of PERCENTILE with the highest winning score. Table 1 shows the tournament results from the point of view of the selective programs.

The current version of LOGISTELLO searches selectively using PERCENTILE = 1.5. In order to gain a further impression of ProbCut's effectiveness, Table 2 lists some statistics about 12–ply searches at various stages of the game averaged over 50 random positions from LOGISTELLO's opening book. Surprisingly, in the opening and midgame phases the accuracy *and* the speed–up are greater than in endgame phase. In many positions the large speed–up enables the selective program to search the relevant lines one or two ply

PERCENTILE ($\Phi \times 100\%$)	Result (Win−Draw−Loss)	Winning Percentage
1.20 (88.5%)	41 − 6 − 23	62.9%
1.35 (91.2%)	45 − 2 − 23	65.7%
1.50 (93.3%)	50 − 4 − 16	74.2%
1.65 (95.1%)	47 − 4 − 19	70.0%
1.80 (96.4%)	49 − 1 − 20	70.7%
2.00 (97.7%)	39 − 6 − 25	60.0%

Table 1: Tournament results: selective vs. non–selective (reported from the point of view of the selective version).

[4] LOGISTELLO's 10–ply evaluation of these positions lies in the range $[-0.2, +0.2]$ which corresponds to winning probabilities in the range $[0.45, 0.55]$.

Disc #	Brute–Force Avg. node #	Std. Dev.	ProbCut(1.5) Avg. node #	Std. Dev.	Speed–Up	Same Move	Same Value
12	5,012,649	1,375,003	752,169	237,304	6.6	100%	84%
20	11,013,030	5,248,987	1,823,702	1,229,339	6.0	90%	72%
28	9,264,054	5,055,533	1,446,873	1,001,035	6.4	100%	90%
36	4,132,991	1,606,994	768,182	424,531	5.4	94%	84%
44	778,925	518,429	181,976	108,519	4.3	84%	78%

Table 2: Comparison of 12 ply searches (50 at each number of discs investigated). The 'Speed–Up' is the ratio of the number of nodes visited; 'Same Move' states the percentage of cases in which both versions selected the same move; 'Same value' refers to the frequency with which both versions reported the same root–position value.

more deeply than the brute–force version, since the average mid–game branching factor for ProbCut searches estimated from Table 2 is less than 3.3. Clearly, the incorporation of ProbCut allows to find decisive moves earlier.

The considerable increase of LOGISTELLO's playing strength can also be gauged by inspecting the result of a tournament between the selective program with 30 minutes thinking time and the non–selective program having 60 minutes per game. For this tournament, the facility enabling the machine 'to think in it's opponent's time' was disabled. The unequivocal result was $41 - 8 - 21$, a winning percentage of 64.3%.

4. Discussion

In this article an easy to implement and memory efficient selective extension of the $\alpha\beta$ algorithm — ProbCut — has been presented which has led to a significant increase of playing strength of a former brute–force Othello program. The idea elaborated in this contribution is that of estimating values returned by deep searches by means of shallow search results so as to reach an early decision whether the deep values fall outside the current search window with a stated probability. Necessary for a successful application of ProbCut is a relatively stable evaluation function or a quiescense search. These properties ensure a small variance of the difference between the true and the estimated evaluation. This makes it possible to cut whole subtrees with confidence very often. ProbCut's applicability is wider than that of the game of Othello, since all game programs with a brute–force kernel tend to examine bad moves to an unmerited depth.

Being aware of the improvements already achieved, one idea to increase the playing strength further is as follows: One weakness of the presented approach is that the depths d' and d are constant. As a result, the average branching factor even when using ProbCut tends to creep up to that without this improvement, and no more further improvements can be expected in comparison with the plain $\alpha\beta$ algorithm by increasing the search depth. To overcome this defect, it is suggested to increase $d - d'$ moderately dependent on the step of the iterative deepening process in order to prune subtrees of increasing depth to keep the average branching factor small.

5. Acknowledgements

I wish to thank all Othello programmers whose programs gave (and give) LOGISTELLO a hard life. Many discussions with these other program's authors favourably influenced my work. Furthermore, thanks go to Warren D. Smith who pointed out the "CHINOOK footnote" after reading my thesis (BURO (1994)) and to the referees for their helpful remarks.

6. References

ANANTHARAMAN, T., CAMPBELL, M.S., HSU, F.–H. (1990). *Singular Extensions: Adding Selectivity to Brute–Force Searching*, ICCA Journal, Vol. 11, No. 4, pp. 135–143. Republished (1990) Artificial Intelligence, Vol. 43, pp. 99–109.

BEAL, D.F. (1990). *A Generalized Quiescence Search Algorithm*, Artificial Intelligence, Vol. 43, pp. 85–98.

BURO, M. (1994). *Techniken für die Bewertung von Spielsituationen anhand von Beispielen*, Ph.D. thesis, University of Paderborn, Germany.

KNUTH, D.E., MOORE, R.W. (1975). *An Analysis of Alpha–Beta Pruning*, Artificial Intelligence, Vol. 6, pp. 293–326.

MCALLESTER, D.A. (1988). *Conspiracy Numbers for MinMax Search*, Artificial Intelligence, Vol. 35, pp. 287–310.

PALAY, A.J. (1985). *Searching with Probabilities*, Pitman Publishing, Originally (1983) published as Ph.D. thesis, Carnegie–Mellon University.

RIVEST, R.L. (1988). *Game Tree Searching by MinMax Approximation*, Artificial Intelligence, Vol. 34, No. 1, pp. 77–96.

SCHAEFFER, J., CULBERSON, J., TRELOAR, N., KNIGHT, B., LU, P., SZAFRON, D. (1992). *A World Championship Caliber Checkers Program*, Artificial Intelligence, Vol. 53, pp. 273–289.