Name: T.A. MARSLAND
Country: CANADA
Affiliation: University of Alberta
Address: Dr. T.A. Marsland,
Computing Science Dept.,
University of Alberta,
EDMONTON T6G 2H1
Alberta, Canada.
Telephone: (403)-432-3971
Technical Area: Computer Software.

Neither this paper nor any version close to it has been or is being offered elsewhere for publication. If accepted, the paper will be presented personally at the 9th World Computer Congress by the author.
A STUDY OF ENHANCEMENTS TO THE ALPHA-BETA ALGORITHM

T.A. Marsland
Computing Science Department
University of Alberta
EDMONTON
08-26-12


ABSTRACT

Data on the relative efficiency of various enhancements to the alpha-beta algorithm is scattered throughout the literature and the results are not always directly comparable. In the present study the performance of new and existing refinements is assessed on a uniform basis. Four enhancements to the alpha-beta algorithm--iterative deepening, aspiration search, memory tables and principal variation search--are compared separately and in various combinations to determine the most effective alpha-beta implementation. Rather than relying on simulation or searches of specially constructed trees, a recently specified data set was analysed by a simple working chess program.
1. INTRODUCTION.

Predicting the outcome of a two-person zero-sum game is equivalent to finding the best sequence of moves in a game tree (i.e. a tree in which the nodes correspond to positions in the game and branches to moves). To determine the best move, the obvious approach is to perform a minimax search of the whole tree. For some complex games, like chess, an exhaustive search is not possible, and so the outcome of the game is approximated by tree searches of some fixed length. When the search algorithm reaches the depth specified, the nodes are considered as terminal, and are subjected to an evaluation function. This function first identifies the non-quiescent moves for special consideration. In the case of chess these are checking or capturing moves. Non-quiescent moves are examined further by building search trees that contain only capturing and checking moves (and their forced responses), until the position becomes quiescent or some maximum depth of search is reached. In contrast, the subtree from each quiescent move at a terminal node is discarded and its value estimated, possibly on a very simple basis of material difference.

The alpha-beta algorithm achieves the same result as minimax, but does so more efficiently. Its approach is to employ two bounds, which form a window. Typically, a call to the alpha-beta function is of the form:

\[ V = AB(p, \alpha, \beta, \text{depth}); \]

where \( p \) is a pointer to a structure which represents a position, \( \alpha \) and \( \beta \) are the lower and upper bounds on the window, and \( \text{depth} \) is the specified length of search. The number returned by the function is called the minimax value of the tree, and measures the
potential success of the next player to move. A skeleton for the
alpha-beta function, expressed in a negamax framework [KNUT75], is
to be found in a recent survey paper [MARS81], where more details
about certain alpha-beta refinements appear. Previous studies of
alpha-beta efficiency have not always been complete, or have been
done on a basis which does not allow for simple comparisons. To
provide more consistency, this new quantitative study presents
results from a simple working chess program¹, and may be compared
with those from searches of specially constructed trees [CAMP82].

2. ALPHA-BETA REFINEMENTS.

The alpha-beta algorithm can take advantage of an iterative
deepening mode, in which a sequence of successively deeper and
deeper searches is carried out until some time limit is exceeded.
Thus a search of depth D ply (moves) may be used to dynamically
reorder (sort) the choices and thus prepare the way for a faster
D+1 ply search than would be possible directly. My aim is to
determine exactly how much a shallow search may improve a deeper
one, and to compare the results with those for a direct full window
search. The methods considered are:
(a). Simple iteration, in which the move list at the root node of
the tree is sorted after each iteration. By this means the
best move found so far is tried first during the next
iteration.
(b). Aspiration search, in which the score returned by the best
move found so far is used as the centre of a narrow window

¹: A 'C' language version of Tinkerbelle [K. Thompson, BTL], a
chess program which participated at the US Computer Chess
within which the score for the next iteration is expected to fall. It is possible for the search to fail, i.e., to return a value which is outside the window. In such a case this partial search may be wasted, although a new centre for the window may be found. Two failure modes are possible: 'low', in which all the moves at the first level (root node) are tried but no value reaches the lower limit of the window, and 'high', upon which the search stops as soon as a move is found which exceeds the upper expectation. A sample implementation of an aspiration search, expressed in the C language with Pascal-style declarations and loops, is shown in Figure 1.

```c
VAR V, e, alpha, beta, D : integer;
p : position;
/* Assume V = estimated value of position p, and 
e = expected error limit. 
Initialize p, depth and e. */
V = 0;
for D = 1 to depth do {
    alpha = V - e;
    beta  = V + e;
    V = AB(p, alpha, beta, D);
    if (V ≥ beta) /* failing high */
        V = AB(p, V, +INF, D);
    else
        if (V ≤ alpha) /* failing low */
            V = AB(p, -INF, V, D);
    sort(p); /* best move so far is tried first on next iteration. */
}
```

Figure 1: Iterative deepening with aspiration search.

Note that +INF corresponds to a value bigger than any that the terminal node evaluation function can produce (e.g., is maxint in Pascal), and that p, depth and e are all presumed to be initialized suitably.
(c). **Minimal window search**, in which it is assumed that the first move to be tried is the start of the **principal variation**. This line is then searched with a full width window, while all the alternate variations are searched with a zero width window, under the assumption that they will fail-low in any case. Should one of the moves not fail this way then it becomes the start of a new principal variation and the search is repeated for this move with a window which covers the new range of possible values.

```pascal
function PVS( p : position; depth : integer) : integer;
{
    VAR width, score, i, value : integer;
    if (depth ≤ 0)                              /* a terminal node? */
        return(evaluate(p));                 /* determine successors p.1 to p.w */
    width = generate(p);                      /* return number of successors */
    if (width == 0)                              /* no legal moves? */
        return(evaluate(p));                /* as a function value */
    make(p.1);
    score = -PVS(p.1, depth-1); /* traverse the PV */
    undo(p.1);
    for i = 2 to width do {
        make(p.i);                            /* try remaining moves */
        value = -AB(p.i, -score-1, -score, depth-1);
        if (value > score)               /* new Principal Variation? */
            score = -AB(p.i, -INF, -value, depth-1);
        undo(p.i);
    }
    return(score);
}
```

Figure 2: Minimal window search.

This method, once referred to as **Calphabeta** [FISH81], will now be called **principal variation search** or **PVS** for short. It is more or less equivalent to **SCOUT** [PEAR81][CAMP82], as shown in Figure 2. Undefined in the program are functions **evaluate** # **generate** # (to assess the value of terminal nodes) # **generate** # (to list
the moves for the current position) \# make \# (to actually play the move considered) and \# undo \# (to retract the current move).

Both aspiration and minimal window searches can be improved by the introduction of memory tables. For this reason the use of refutation and transposition tables forms a part of the study.

3. MEMORY TABLES.

After a search to depth $D$ on a tree of constant width $W$ a refutation table will contain $W \times D$ entries. For each variation the sequence of $D$ moves which determined a sufficient value (cut off the search) for that variation is stored in the table. Prior to the next iteration the table is sorted so that the new candidate principal variation is tried first. Thus on an iteration to depth $D+1$ there exists a $D$-ply sequence that is tried immediately. The next ply is then added and the search continues. The candidate principal variation is fully searched, but for the alternate variations the moves in the refutation table may be sufficient to cut off the search again and thus save the move generation that would normally occur at each node. The storage overhead is very small, although a small triangular table is also needed to identify the refutations [AKL77].

A transposition table may also be used to hold refutations but, because it has the capacity for including more information, it has other capabilities too. In Figure 3 a tree of constant width $W = 3$ and uniform depth $D = 3$ is represented. The positions actually stored in the table are shown by the solid lines. The branches with solid or double dot lines are actually searched by the alpha-beta algorithm, while those with single dots are not
searched at all, i.e., are cut off.

Figure 3: 3-ply tree showing transposition table entries.

The numbers at the terminal nodes are produced by an evaluation function. The other numbers are the values of the individual subtrees, as passed back (backed up) to the root node by the alpha-beta process. From this one can see that the minimax value of the tree is 4 and that the results from 15 positions would be stored, rather than only 9 in the refutation table case. Thus the transposition table contains not only the main line of each variation but also the main subvariations. If the information stored in the entries contains at least the best move in the position and the value and length of the subtree emanating from that point, then the transposition table may also be used to extend the effective search depth [MARS81]. This is especially valuable in endgames when the number of possible alternatives is small. As in the other cases, a sorting operation between each iteration ensures that the moves at the first level will be tried in the best possible order. A typical transposition table might contain 10,000 entries, each of 10 bytes [MARS81], for a 100,000 byte total storage overhead.
4. BASIS FOR COMPARISON.

In comparing algorithms which search game trees, two basic criteria are employed. One may either measure the amount of computer time used to search a tree, the method which consistently produces the expected result in least time being superior, or one may count the number of nodes visited in the tree. If the cost of a node is nearly constant, these two measures are effectively the same. However, the test program, and chess programs in general, perform significantly more calculation at terminal nodes than at interior nodes in the tree, since they carry out a check or capture analysis in the form of an extended tree search. Therefore the following results are based on the number of terminal nodes examined, especially since this provides a machine-independent measure for future comparisons.

5. RESULTS.

The algorithms were tested on a data set which was used to assess the performance of computer chess programs and human players [BRAT82]. That data set contained 24 chess positions [MARS82], of which one was deleted since it involved a simple sequence of forcing checks. All the remaining positions were searched with 3, 4 and 5-ply trees, using a combination of alpha-beta refinements, and a 6-ply search was done with best method. The raw results have been condensed into two graphs. Because the number of terminal nodes is exponential with the depth of search, the average terminal node count per position is plotted on a log-linear graph, Figure 4. The results give a good indication of the relative merits of each alpha-beta refinement. However, the effectiveness of the various methods is perhaps better seen in Figure 5, which shows the ratio
of the number of terminal nodes searched relative to a direct search. From the graphs one may also deduce that for our data the incremental cost, using iterative deepening, of an odd ply search after an even ply one is approximately twice as large as the incremental cost of an even ply search. This result agrees with the earlier ones of Gillogly [GILL72] and Slagle [SLAG69], even though those studies were for direct alpha-beta searches, that is to say, did not include transposition table and other enhancements.

Since a transposition table is accessed like a hash table, its usage is most effective if the initial probes are uniformly distributed across all the table entries. If there is a conflict, that is, if the initial entry contains valid data but is not the one sought, then a sequence of secondary entries may be tried. The maximum acceptable length of this sequence is an important parameter. It is recognized that an exhaustive search of the whole table may be too time-consuming. So, for example, in BLITZ\(^2\) a secondary sequence length of ten is used, while in BELLE\(^3\) only the initial entry is considered. The latter approach was adopted here because it is simpler, even though the 8192-entry transposition table was comparatively small. Our results indicate that determining the most effective way to use a transposition table is very important, since it is clear from Figure 4 that there is considerable scope for improvement in these algorithms, especially in the even ply cases.

In order to provide a lower bound on the number of terminal nodes for our chosen data set, it is necessary to estimate the

\(^2\) BLITZ, a master calibre chess program developed by R. Hyatt, Univ. of Southern Mississippi.
\(^3\) BELLE, the current world champion chess program, developed by K. Thompson, Bell Laboratories.
minimal tree that must be searched by the alpha-beta algorithm. If we assume that these game trees may be modelled by a uniform tree of constant width $W$, and that $W$ may be estimated by computing the number of branches divided by the number of nodes in the actual game tree, then the average of these estimates may be taken as the constant width of a representative tree. On trees of constant width $W$ and fixed depth $D$, there is a formula for the minimal size of the tree that must be searched by the alpha-beta algorithm, and it is given by the expression

\[
W^{\lceil D/2 \rceil} + W^{\lfloor D/2 \rfloor} - 1 \text{ nodes} \quad [\text{SLAG69}],
\]

where $\lceil x \rceil$ and $\lfloor x \rfloor$ represent upper/lower integer bounds on $x$.

We have plotted the minimal search size under optimal conditions in Figures 4 & 5, and one can see that a factor of 1.2 reduction is possible on 3 and 5-ply trees and a factor of about 2.5 on 4 and 6-ply trees. The true reason for this difference is not clear, although factors of two between even and odd ply searches are common. On the other hand, perhaps the data set of 23 positions is too small or is biased in some way. In fact, one of the positions does influence the final results strongly. For example, in the case of board $W$ a change occurred in the principal variation, thus the 4-ply search was not a good predictor of the 5-ply result. Just how serious this can be is clear from Table 1 which shows that for board $W$ all the iterative searches are more expensive than a direct search. This is reinforced in the 6-ply results when, for the case PVS with transposition table, 28% of the effort was expended on board $W$ (Table 2).

Although the efficiency of the various methods changed when done on a CPU basis, rather than on a terminal node count, the
relative efficiency of the methods was not affected. While one may argue that the terminal node count does not reflect the true cost of a search, it does make possible a direct comparison with the expected minimal tree size (Figure 4).

6. CONCLUSIONS.

These results confirm that iterative deepening is an effective enhancement to the alpha-beta algorithm, provided it is used in conjunction with some form of aspiration or memory table search. For relatively shallow trees (depth \( \leq 5 \)) there is not much to choose between refutation and transposition memory tables. By its very nature, a transposition table is continually being filled with new positions, some of which may destroy entries that have not yet been reused. Thus it is not possible to guarantee that all the primary refutations will be retained. A refutation table does not suffer from this problem and, since it is small and easy to maintain, it is recommended that it always supplement a transposition table, thus guaranteeing retention of the primary refutations. In our experience, the combination memory function is, on average, measurably better than use of a transposition table alone. To support this combination we observed that, for the 5-ply PVS case an average 2 percentage point improvement occurred, while in the 6-ply case (Table 2) a more dramatic 31 percentage point improvement is shown. From this second result we conclude that a transposition table of 8192 entries is too small for 6-ply searches of complex positions, since it becomes seriously overcommitted and cannot perform as well as the simpler refutation table. On the other hand, the true power of a transposition table was not brought out by our data set, since there were only two endgames, boards F
and H (Tables 1 & 2).

Two interesting problems arose during this study. Computation of the (hash) transposition table key was based on the method described by Zobrist [ZOBR70]. During the 6-ply searches this was found to be inadequate, when two different positions generated the same hash-key code. The difficulty was eased by extending the codes, which represent placement of pieces on the board, from 32 to 48 bits, thus decreasing the probability of such a conflict. A second problem arose from a subtle inconsistency between refutation table and transposition table usage. This was corrected by abandoning the refutation sequence as soon as the transposition table offered an alternative. It is easy for this problem to occur when enpassant captures exist, since the same position can arise from two different move sequences. A simpler way of maintaining consistency is to give the refutation table entries priority over those in the transposition table. Since replications of alternate variations occur infrequently, the only penalty is an insignificant loss of efficiency.

Of the two principal refinements, narrow or minimal window aspiration search and memory tables, it was found that preservation and use of the refutations from a previous iteration was more important than aspiration searching. This point is clearly illustrated in Table 1, where a full window search with refutation table support is superior to a narrow window aspiration search without memory table. In general, although the data in Table 1 appears to support that possibility, the combined effect of these two refinements is not additive, but improvement in performance occurs, especially for the deeper searches.

Based on our experiments, as summarized by results presented
in Figure 5, it is clear that PVS is potentially superior to narrow window aspiration searching, and avoids the need to determine the optimal window size. Note that this result is contrary to an earlier conclusion for the game of checkers [FISH81], where Calphabeta (that is, PVS) was described as being "disappointing" and "probably not to be recommended" [FISH81]. Thus for two different games contradictory results appear, illustrating how game-dependent these methods may be and the importance of strong move ordering [MARS81] in the efficiency of tree search algorithms.
REFERENCES


<table>
<thead>
<tr>
<th>Board</th>
<th>Full Window</th>
<th>Direct</th>
<th>Iterative</th>
<th>No Table</th>
<th>Refutation Table</th>
<th>Transposition Table</th>
<th>Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>61773</td>
<td>68399</td>
<td></td>
<td>46732</td>
<td>50625</td>
<td>69198</td>
<td>46845</td>
</tr>
<tr>
<td>B</td>
<td>50861</td>
<td>57519</td>
<td></td>
<td>34332</td>
<td>41019</td>
<td>34208</td>
<td>28227</td>
</tr>
<tr>
<td>C</td>
<td>58622</td>
<td>59437</td>
<td></td>
<td>55549</td>
<td>54294</td>
<td>50398</td>
<td>49370</td>
</tr>
<tr>
<td>D</td>
<td>180659</td>
<td>196349</td>
<td></td>
<td>94730</td>
<td>97074</td>
<td>111465</td>
<td>88807</td>
</tr>
<tr>
<td>E</td>
<td>24645</td>
<td>27364</td>
<td></td>
<td>20285</td>
<td>14151</td>
<td>26162</td>
<td>19472</td>
</tr>
<tr>
<td>F</td>
<td>116933</td>
<td>136416</td>
<td></td>
<td>84855</td>
<td>75801</td>
<td>94992</td>
<td>65194</td>
</tr>
<tr>
<td>G</td>
<td>7612</td>
<td>9116</td>
<td></td>
<td>8253</td>
<td>6124</td>
<td>5481</td>
<td>5108</td>
</tr>
<tr>
<td>H</td>
<td>132306</td>
<td>144505</td>
<td></td>
<td>86565</td>
<td>80933</td>
<td>81554</td>
<td>66957</td>
</tr>
<tr>
<td>I</td>
<td>181883</td>
<td>192933</td>
<td></td>
<td>112237</td>
<td>104027</td>
<td>127312</td>
<td>80331</td>
</tr>
<tr>
<td>J</td>
<td>109371</td>
<td>119427</td>
<td></td>
<td>56635</td>
<td>62999</td>
<td>65390</td>
<td>52342</td>
</tr>
<tr>
<td>K</td>
<td>78580</td>
<td>82932</td>
<td></td>
<td>43260</td>
<td>53514</td>
<td>53708</td>
<td>38600</td>
</tr>
<tr>
<td>L</td>
<td>143048</td>
<td>152922</td>
<td></td>
<td>139816</td>
<td>92164</td>
<td>111316</td>
<td>107346</td>
</tr>
<tr>
<td>M</td>
<td>31812</td>
<td>31701</td>
<td></td>
<td>31418</td>
<td>29875</td>
<td>30573</td>
<td>30273</td>
</tr>
<tr>
<td>N</td>
<td>34092</td>
<td>27048</td>
<td></td>
<td>25084</td>
<td>23459</td>
<td>22788</td>
<td>22225</td>
</tr>
<tr>
<td>O</td>
<td>75841</td>
<td>56372</td>
<td></td>
<td>51801</td>
<td>42900</td>
<td>50007</td>
<td>48075</td>
</tr>
<tr>
<td>P</td>
<td>85844</td>
<td>91284</td>
<td></td>
<td>72159</td>
<td>62378</td>
<td>51742</td>
<td>41842</td>
</tr>
<tr>
<td>Q</td>
<td>188877</td>
<td>201361</td>
<td></td>
<td>18809</td>
<td>12856</td>
<td>142188</td>
<td>134292</td>
</tr>
<tr>
<td>R</td>
<td>65370</td>
<td>82351</td>
<td></td>
<td>47504</td>
<td>52128</td>
<td>71536</td>
<td>43645</td>
</tr>
<tr>
<td>S</td>
<td>26078</td>
<td>287118</td>
<td></td>
<td>224568</td>
<td>171356</td>
<td>97785</td>
<td>78942</td>
</tr>
<tr>
<td>T</td>
<td>257810</td>
<td>223869</td>
<td></td>
<td>152228</td>
<td>124901</td>
<td>138113</td>
<td>107773</td>
</tr>
<tr>
<td>U</td>
<td>54032</td>
<td>64938</td>
<td></td>
<td>51318</td>
<td>45695</td>
<td>49705</td>
<td>43828</td>
</tr>
<tr>
<td>V</td>
<td>142147</td>
<td>307806</td>
<td></td>
<td>275530</td>
<td>212299</td>
<td>229395</td>
<td>192615</td>
</tr>
<tr>
<td>W</td>
<td>68567</td>
<td>73174</td>
<td></td>
<td>68008</td>
<td>71768</td>
<td>69627</td>
<td>67835</td>
</tr>
<tr>
<td>X</td>
<td>104390</td>
<td>117122</td>
<td></td>
<td>85690</td>
<td>73828</td>
<td>77312</td>
<td>63459</td>
</tr>
<tr>
<td>Total</td>
<td>241476</td>
<td>269382</td>
<td></td>
<td>197086</td>
<td>169804</td>
<td>177813</td>
<td>145957</td>
</tr>
</tbody>
</table>

Table 1: 5-ply terminal node count for alpha-beta variations.
<table>
<thead>
<tr>
<th></th>
<th>full window</th>
<th>transposition table</th>
<th>transposition and refutation table</th>
<th>move</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(forced mate)</td>
<td>118055</td>
<td>92776</td>
<td>d6d1</td>
</tr>
<tr>
<td>B</td>
<td>157843</td>
<td>118055</td>
<td>92776</td>
<td>e4e5</td>
</tr>
<tr>
<td>C</td>
<td>270258</td>
<td>578855</td>
<td>130859</td>
<td>e8d8</td>
</tr>
<tr>
<td>D</td>
<td>100498</td>
<td>151945</td>
<td>96232</td>
<td>e5e6</td>
</tr>
<tr>
<td>E</td>
<td>502855</td>
<td>367191</td>
<td>347057</td>
<td>a1d1</td>
</tr>
<tr>
<td>F</td>
<td>48980</td>
<td>30875</td>
<td>27743</td>
<td>g5g6</td>
</tr>
<tr>
<td>G</td>
<td>552347</td>
<td>499806</td>
<td>410734</td>
<td>h5f6</td>
</tr>
<tr>
<td>H</td>
<td>26314</td>
<td>10985</td>
<td>9236</td>
<td>a2a3</td>
</tr>
<tr>
<td>I</td>
<td>547563</td>
<td>316397</td>
<td>272255</td>
<td>c3b5</td>
</tr>
<tr>
<td>J</td>
<td>100498</td>
<td>151945</td>
<td>96232</td>
<td>e5e6</td>
</tr>
<tr>
<td>K</td>
<td>502855</td>
<td>367191</td>
<td>347057</td>
<td>a1d1</td>
</tr>
<tr>
<td>L</td>
<td>48980</td>
<td>30875</td>
<td>27743</td>
<td>g5g6</td>
</tr>
<tr>
<td>M</td>
<td>552347</td>
<td>499806</td>
<td>410734</td>
<td>h5f6</td>
</tr>
<tr>
<td>N</td>
<td>26314</td>
<td>10985</td>
<td>9236</td>
<td>a2a3</td>
</tr>
<tr>
<td>O</td>
<td>547563</td>
<td>316397</td>
<td>272255</td>
<td>c3b5</td>
</tr>
<tr>
<td>P</td>
<td>100498</td>
<td>151945</td>
<td>96232</td>
<td>e5e6</td>
</tr>
<tr>
<td>Q</td>
<td>502855</td>
<td>367191</td>
<td>347057</td>
<td>a1d1</td>
</tr>
<tr>
<td>R</td>
<td>48980</td>
<td>30875</td>
<td>27743</td>
<td>g5g6</td>
</tr>
<tr>
<td>S</td>
<td>552347</td>
<td>499806</td>
<td>410734</td>
<td>h5f6</td>
</tr>
<tr>
<td>T</td>
<td>26314</td>
<td>10985</td>
<td>9236</td>
<td>a2a3</td>
</tr>
<tr>
<td>U</td>
<td>547563</td>
<td>316397</td>
<td>272255</td>
<td>c3b5</td>
</tr>
<tr>
<td>V</td>
<td>100498</td>
<td>151945</td>
<td>96232</td>
<td>e5e6</td>
</tr>
<tr>
<td>W</td>
<td>502855</td>
<td>367191</td>
<td>347057</td>
<td>a1d1</td>
</tr>
<tr>
<td>X</td>
<td>48980</td>
<td>30875</td>
<td>27743</td>
<td>g5g6</td>
</tr>
<tr>
<td>Total</td>
<td>10227550</td>
<td>10018219</td>
<td>5803333</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>444676</td>
<td>435574</td>
<td>252318</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>100</td>
<td>100</td>
<td>57</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 2: 6-ply search data, node count and time.