RELATIVE EFFICIENCY OF ALPHA-BETA IMPLEMENTATIONS

T.A. Marsland
Computing Science Department
University of Alberta
EDMONTON T6G 2H1
Canada

ABSTRACT

Most of the data on the relative efficiency of different implementations of the alpha-beta algorithm is neither readily available nor in a form suitable for easy comparisons. In the present study 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. The rationale for this work is to ensure that new parallel algorithms incorporate the best sequential techniques. Rather than relying on simulation or searches of specially constructed trees, a simple chess program was used to provide a uniform basis for comparisons.

I INTRODUCTION

Perhaps the most complete description of the alpha-beta algorithm is the paper by Knuth and Moore, in which a negamax implementation is described [1]. That paper also makes a clear distinction between those nodes in the game tree where cutoffs may occur, and those which must be fully explored, and so are logical candidates for the application of multiple processors during parallel searches. One field where the alpha-beta algorithm is universally applied is that of computer chess. Here the problems are so large that a tree of the whole game cannot be built and so an approximate solution is sought, one which involves a succession of searches on fixed depth trees. At a terminal node (a leaf) an evaluation function is invoked to estimate the value of the subtrees discarded. In chess, non-quietly moves at the terminal nodes are explored more fully, with special subset searches involving, for example, only moves which check or capture (or their forced responses).

The alpha-beta algorithm owes its efficiency to the employment of two bounds which form a window. Typically, a call to the alpha-beta function is of the form:

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

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 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 appears in a recent survey paper [2], where more details about certain alpha-beta refinements appear. Previous studies of alpha-beta efficiency have not considered these refinements, or have not been done on a basis which allows for simple comparisons. To provide more consistency, this new quantitative study presents results from a simple working chess program, and may be compared with those from searches of specially constructed trees [3].

II ALPHA-BETA REFINEMENTS

An iterative deepening mode, in which a sequence of successively deeper and deeper searches is carried out until some time limit is exceeded, is a simple way of extending the alpha-beta algorithm. A search of depth D ply (moves) is used to dynamically reorder (sort) the choices and thus prepare the way for a faster search to D+1 ply 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:

1. Simple iteration, in which the move list at the root node of the tree is sorted after each iteration. By this means the candidate best move is tried first during the next iteration.

2. Aspiration search, in which the score returned by the best move found so far is used as the centre of a narrow window within which the score for the next iteration is expected to fall. If the value returned is outside the window, the search has failed high or low and must be repeated with a window which spans the new range of possible values [2].

3. Minimal window search employs a full window only on the candidate principal variation. All the alternate variations are searched with a zero 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 correct range of possible values. The PVS (principal variation search) implementation of this algorithm is based on Calphabeta [4], which in turn is similar.

lar to Scout [5]. The algorithm is presented in Figure 1, through a Pascal like language extended with a return statement. Undefined in the program are functions evaluate (to assess the value of a leaf), and generate (to list the moves for the current position). For simplicity, additional functions make (to actually play the move considered) and undo (to retract the current move) are not included. Note that PVS preserves the property of Falphabeta [4], in that for failing searches the bound returned may be better than the alpha limit. This means that the re-search of a new principal variation normally proceeds with a narrower window. More importantly, PVS may be easily extended to draw on the idea that the correct score for a candidate principal variation is not needed until a potential rival arises. This extension to alphabeta searching is based on a technique employed by K. Thompson at the first level of the tree search in Belle 1. Note also that zero window searches normally cut off quite quickly. If this is not the case, then the profitable heuristic is to curtail the search and repeat immediately with the appropriate window.

Naturally, all of these methods may be improved by the inclusion of transposition and refutation memory tables.

III MEMORY TABLES

For each initial move in the game tree, the alpha-beta algorithm determines a sequence of moves which is sufficient to cut off the search. These sequences may be stored in a refutation table. After a search to depth D on a tree of constant width W this table will contain W*D entries. Thus upon the next iteration there exists a set of move sequences of length D-ply that are to be tried first. 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 again be sufficient to cut off the search, 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 [6].

A transposition table holds not only refutations and the main subvariations, but also has the capacity for including more information. In particular, once a subtree has been searched its transposition table entry will contain not only the length of the search tree and the value of the subtree, but also whether that value represents the true score or an upper/lower bound on the score [2]. A typical transposition table might contain 100,000 entries, each of 10 bytes, for a million-byte total storage overhead. In our implementation, the (position encoding) hash key was 48 bits long, of which 12 bits were used to index into an 8192-entry table. Various choices for accessing the transposition table are discussed in a recent report [7]. For this study only a single probe of the table was made for each position.

IV RESULTS

Minimax tree searches generally involve significantly more calculation at a leaf than at an interior node. For example, chess programs carry out a check and capture analysis in the form of an extended tree search. Therefore the following results are based on the number of terminal nodes examined. It is reasonable to assume that the various heuristics in the evaluation function are equally effective across all alpha-beta refinements, and so we have a machine-independent measure for future comparisons.

The algorithms were tested on a data set which was used to assess the performance of computer chess programs and human players [8]. That data set contains 24 chess positions (labelled A...X in Table 1), but A was deleted from our study 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 alphabeta refinements, and a 6-ply search was done with the best method. The raw results have been condensed into Figure 2, which shows the ratio of the number of terminal nodes searched relative to a direct search. In order to see how much improvement is possible in the alpha-beta algorithm, the formula

\[ W^{*}[D/2] + W^{*}[D/2] - 1 \text{ nodes,} \]

where \([x]\) and \([x]\) represent upper/lower integer bounds on x, is plotted in Figure 2 as the minimum tree size [9]. Here the value for W is estimated as the average width of the nodes in the trees being studied. The zig-zag appearance of Figure 2 is normal for alpha-beta searches [10], and occurs because for an even-ply search a larger fraction of the terminal nodes must be fully evaluated.

From Table 1 we see that one of the positions influences 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, 26% of the effort was expended on board W [7]. Some effective heuristics for partial re-ordering of the move list between iterations can be developed to correct this problem. Even so, iterative searches may be at a disadvantage whenever the principal variation changes. For problems of this type we are designing parallel versions of PVS.

V ASSESSMENT OF SEQUENTIAL METHODS

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 ≤ 5) there is not much to choose between refutation and transposition memory tables. By its very nature, a transposition table is continually being filled.
of the two principal refinements, narrow window aspiration search and use of 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 a memory table.

Based on our experiments, as summarized by results presented in Figure 2, it is clear that PVS is potentially superior to narrow window aspiration searching, since it avoids the need to determine an acceptable window. Note that these results reverse an earlier conclusion for the game of checkers, where Calphastrosp was described as being "disappointing" and "probably not to be recommended" [4]. Thus for two different games, contradictory results appear, illustrating not only how game-dependent these methods may be, but also the influence of strong move ordering [2] on the efficiency of tree search algorithms.

REFERENCES


### Table 1: 5-ply terminal node count for alpha-beta variations

<table>
<thead>
<tr>
<th>Board</th>
<th>full window direct table</th>
<th>iterative table</th>
<th>no tables asp</th>
<th>PVS</th>
<th>refutation table asp</th>
<th>PVS</th>
<th>transposition table asp</th>
<th>PVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>46732</td>
<td>50625</td>
<td></td>
<td></td>
<td>69198</td>
<td>46485</td>
<td>48196</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>50861</td>
<td>97539</td>
<td></td>
<td></td>
<td>34208</td>
<td>28927</td>
<td>30914</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>89422</td>
<td>98347</td>
<td></td>
<td></td>
<td>50398</td>
<td>48370</td>
<td>48410</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>198659</td>
<td>96349</td>
<td></td>
<td></td>
<td>111465</td>
<td>88607</td>
<td>88125</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>256869</td>
<td>90705</td>
<td></td>
<td></td>
<td>26162</td>
<td>94782</td>
<td>14020</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>116932</td>
<td>13616</td>
<td></td>
<td></td>
<td>94992</td>
<td>65184</td>
<td>60817</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>56712</td>
<td>9116</td>
<td></td>
<td></td>
<td>5481</td>
<td>5108</td>
<td>4706</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>172206</td>
<td>149505</td>
<td></td>
<td></td>
<td>81554</td>
<td>66857</td>
<td>67232</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>181883</td>
<td>192929</td>
<td></td>
<td></td>
<td>127312</td>
<td>80323</td>
<td>80874</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>109371</td>
<td>11427</td>
<td></td>
<td></td>
<td>65390</td>
<td>52322</td>
<td>51924</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>78590</td>
<td>8358</td>
<td></td>
<td></td>
<td>53708</td>
<td>38600</td>
<td>44420</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>134089</td>
<td>152282</td>
<td></td>
<td></td>
<td>111316</td>
<td>107346</td>
<td>85779</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>32162</td>
<td>31701</td>
<td></td>
<td></td>
<td>11961</td>
<td>49200</td>
<td>39834</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>164092</td>
<td>56377</td>
<td></td>
<td></td>
<td>18329</td>
<td>106427</td>
<td>43502</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>43941</td>
<td>33584</td>
<td></td>
<td></td>
<td>99795</td>
<td>78842</td>
<td>74216</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>175810</td>
<td>223869</td>
<td></td>
<td></td>
<td>142188</td>
<td>124292</td>
<td>97861</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>54032</td>
<td>64938</td>
<td></td>
<td></td>
<td>138113</td>
<td>107733</td>
<td>96104</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>142147</td>
<td>307806</td>
<td></td>
<td></td>
<td>127312</td>
<td>80323</td>
<td>80874</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>86597</td>
<td>73174</td>
<td></td>
<td></td>
<td>94992</td>
<td>65184</td>
<td>60817</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>2414763</td>
<td>2693821</td>
<td></td>
<td></td>
<td>1778183</td>
<td>1459574</td>
<td>1362856</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>102796</td>
<td>101382</td>
<td></td>
<td></td>
<td>94992</td>
<td>65184</td>
<td>60817</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>100</td>
<td>111</td>
<td></td>
<td></td>
<td>77311</td>
<td>62459</td>
<td>58254</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>100</td>
<td>111</td>
<td></td>
<td></td>
<td>77311</td>
<td>62459</td>
<td>58254</td>
<td></td>
</tr>
</tbody>
</table>

Number of Terminal Nodes Evaluated (5-ply)
FUNCTION PVS (p : position; alpha, beta, depth : integer) : integer;
VAR width, score, i, value, bound : integer;
BEGIN
    IF (depth = 0) THEN
        return(evaluate(p));
    { determine successors p.1 to p.w }
    width := generate(p);
    { return number of successors }
    IF (width = 0) THEN
        return(evaluate(p));
    { leaf, no moves? }
    score := -PVS(p.1, -beta, -alpha, depth-1);
    IF (score < beta) THEN
        { no cutoff }
        FOR i := 2 TO width DO BEGIN
            bound := MAX(score, alpha);
            value := -PVS(p.i, -bound-1, -bound, depth-1);
            IF (value > score) THEN
                score := -PVS(p.i, -beta, -value, depth-1);
            IF (score ≥ beta) THEN
                return(score);
        END (for loop);
        return(score);
    END (PVS);

Figure 1: Depth-Limited Principal Variation Search.

Figure 2: Performance Comparison of Alphabeta Enhancements.

KEY
- Simple iteration, full window.
+ Direct search, full window.
□ Narrow window, no tables.
⊙ Full window, refutation table.
× PVS, transposition table.
⊗ PVS, transposition and refutation tables.
※ Estimated minimal tree.