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-quiescent 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:

```
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[1], 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 simi-

---

1: Tinkerbelle [K. Thompson], a chess program which participated at the US Computer Chess Championship, San Diego, November, 1974.

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 alpha-beta searching is based on a technique employed by K. Thompson at the first level of the tree search in Belle[2]. Note also that zero window searches normally cut off quite quickly. If this is not the case, then a 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.

2: BELLE, the current world champion chess program, developed by K. Thompson, Bell Laboratories.

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 alpha-beta 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

$$W^{**}\lceil D/2 \rceil + W^{**}\lfloor D/2 \rfloor - 1 \text{ nodes,}$$

where $\lceil x \rceil$ and $\lfloor x \rfloor$ 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 occur 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, 28% 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 ith new positions, some of which may destroy en-ries that have not yet been reused. Thus it is ot possible to guarantee that all the primary efutations will be retained. This problem can be vercome through the inclusion of a small and easy o maintain refutation table. To support this com-ination, we observed that for the 5-ply PVS case n average 2 percentage point improvement oc-curred, while in the 6-ply case a more dramatic 31 percentage point improvement was seen, Figure 2. From this second result we conclude that a trans-position table of 8192 entries is too small for 6-ply searches of complex positions, since it be-comes 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 (Table 1).

VI CONCLUSIONS

Of the two principal refinements, narrow window aspiration search and use of memory tables, it was found that preservation and use of the re-futations 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 wi-thout 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 as-piration 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 Calphabeta 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

[1] Knuth, D. and R. Moore, "An Analysis of Alpha-beta Pruning". Artificial Intelligence 6 (1975) 293-326.

[2] Marsland, T.A. and M. Campbell, "Parallel Search of Strongly Ordered Game Trees". ACM Computing Surveys 14:4 (1982) 533-551.

[3] Campbell, M.S. and T.A. Marsland, "A Comparison of Minimax Tree Search Algorithms". Artificial Intelligence (1983).

[4] Fishburn, J. "Analysis of Speedup in Distributed Algorithms", Ph.D. thesis, TR #431, Computer Sciences Dept., Univ. of Wisconsin, Madison, May 1981.

[5] Pearl, J. "Asymptotic properties of minimax trees and game searching procedures". Artificial Intelligence 14 (1980) 113-138.

[6] Akl, S.G. and M.M. Newborn, "The Principal Continuation and the Killer Heuristic". In Proc. ACM National Conf., Seattle, October, 1977, pp. 466-473.

[7] Marsland, T.A. "A Quantitative Study of Refinements to the Alpha-beta Algorithm", TR82-6, Comp. Sci. Dept., Univ. of Alberta, Edmonton, August, 1982.

[8] Bratko, I. and D. Kopec, "A Test for Comparison of Human and Computer Performance in Chess". Advances in Computer Chess 3, M.R.B. Clarke (editor), Pergamon Press, 1982, pp. 57-69.

[9] Slagle, J.R. and J.K. Dixon, "Experiments with some Programs which Search Game Trees". Journal ACM 16:2 (1969) 189-207.

[10] Gillogly, J.J. "The Technology Chess Program". Artificial Intelligence 3 (1972) 145-163.

Number of Terminal Nodes Evaluated (5-ply)										
board	full window		no tables		refutation table			transposition table	
	direct	iterative	asp	PVS	full	asp	PVS	asp	PVS
A	(forced	mate)							
B	61773	68399	46732	50625	69198	46485	48196	44052	46810
C	50861	57539	34332	41019	34208	28227	30484	27300	30275
D	58622	59437	55549	54294	50398	49370	48410	47226	47151
E	180659	196349	94730	97074	111465	88807	88125	84515	84068
F	24645	27364	20285	14151	26162	19472	14020	12579	12413
G	116933	136416	84855	75801	94992	65194	60817	62586	57342
H	7612	9116	8253	6124	5481	5108	4706	4086	4107
I	132306	144505	86565	80933	81554	66957	67822	62556	67150
J	181883	192933	112237	104027	127312	80331	80974	84774	79273
K	109371	119427	56635	62999	65390	52342	51954	48968	48772
L	78580	82392	43260	53514	53708	38600	44420	35853	38661
M	143048	152922	139816	92164	111316	107346	85779	89234	82629
N	31812	31701	31418	29875	30573	30273	29834	29694	29664
O	34092	27048	25084	23459	22788	22225	21550	21652	21528
P	75841	56372	51801	42900	50007	48075	40102	40518	39647
Q	85844	91284	72159	62378	51742	41842	37859	33933	33924
R	188877	201361	188009	128565	142188	134292	97861	94138	87243
S	65370	82351	47504	52128	71536	43645	43762	41197	41512
T	264078	287118	224568	171356	97785	78942	74026	130266	92728
U	257810	223869	152228	124901	138113	107773	96104	99303	94603
V	54032	64938	51318	45695	49705	43818	41810	39644	39178
W	142147	307806	275530	212299	222935	192615	186438	179855	159550
X	68567	73174	68008	71768	69627	67835	67803	67514	67515
Total	2414763	2693821	1970876	1698049	1778183	1459574	1362856	1381443	1305743
Mean	104990	117122	85690	73828	77312	63459	59254	60062	56771
%	100	111	82	70	74	60	56	57	54

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

```
FUNCTION PVS (p : position; alpha,beta,depth : integer) : integer;
  VAR width, score, i, value, bound : integer;
BEGIN                               { assert depth positive }
  IF (depth = 0) THEN               { leaf, maximum depth? }
     return(evaluate(p));
                                    { determine successors p.1 to p.w }
  width := generate(p);             { return number of successors      }
                                    { as the function value            }
  IF (width = 0) THEN               { leaf, no moves? }
     return(evaluate(p));

  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                  { cutoff? }
        return(score);
  END {for loop};
  return(score);
END {PVS};
```

Figure 1: Depth-Limited Principal Variation Search.

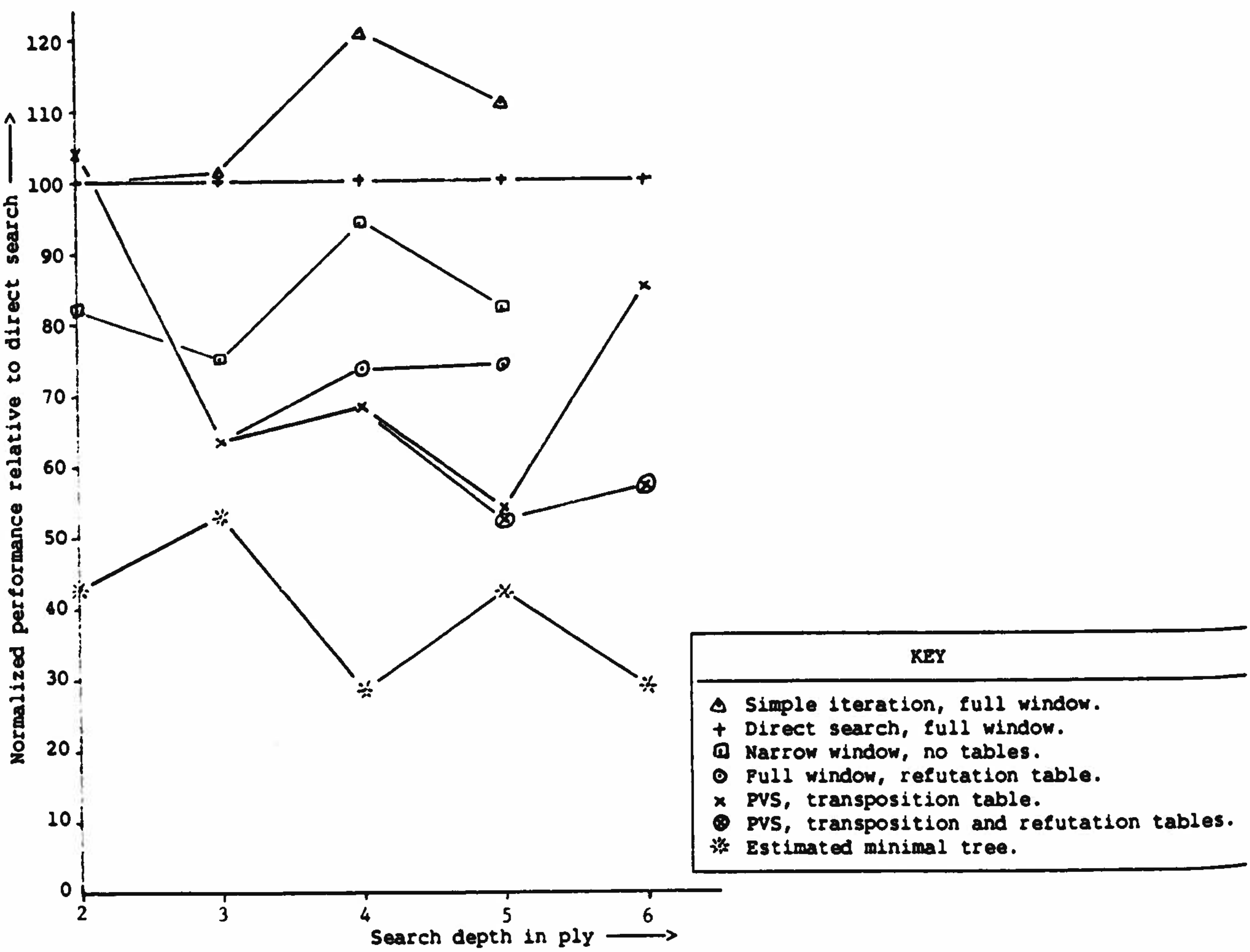

Figure 2: Performance Comparison of Alphabeta Enhancements.