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

Draft: IFIP'83 Conference, Paris.

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, 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, 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 program[1], 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 <u>ply</u> (moves) may be used to <u>dynamically reorder</u> (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 <u>full window</u> search. The methods considered are:

(a). <u>Simple iteration</u>, 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). <u>Aspiration search</u>, 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 Championship, ACM National Conference, San Diego, 1975.

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.

```
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.

```
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     */
                              /* as a function value             */
  if (width == 0)                                /* no legal moves? */
     return(evaluate(p));

  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 # (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*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.

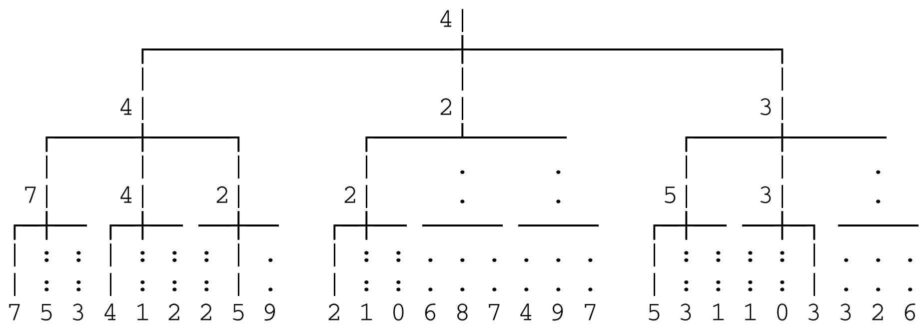

Figure 3: 3-ply tree showing transposition table entries.

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 <u>uniform</u> 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

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

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 ≤ 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 abandonning 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

AKL77 S.G. Akl and M.M. Newborn, "The Principal Continuation and the Killer Heuristic", Proc. ACM National Conf., Seattle 1977, 466-473.

BRAT82 I. Bratko 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.

CAMP82 M.S. Campbell and T.A. Marsland, "A Comparison of Minimax Tree Search Algorithms", TR82-3, University of Alberta, (to appear in Artificial Intelligence--early 1983).

FISH81 J. Fishburn, "Analysis of Speedup in Distributed Algorithms", TR #431, Computer Sciences Dept., University of Wisconsin-Madison, 1981.

GILL72 J.J. Gillogly, "The Technology Chess Program", Artificial Intelligence 3 (1972), 145-163.

KNUT75 D. Knuth and R. Moore, "An Analysis of Alpha-beta Pruning", Artificial Intelligence 6 (1975), 293-326.

MARS81 T.A. Marsland and M. Campbell, "Parallel Search of Strongly Ordered Game Trees", TR81-9, University of Alberta, revised June 1982, (to appear ACM Computing Surveys, Dec. 1982).

MARS82 T.A. Marsland, "A Quantitative Study of Refinements to the Alpha-beta Algorithm", TR82-6, Comp. Sci. Dept., Univ. of Alberta, Aug. 1982.

PEAR80 J. Pearl, "Asymptotic Properties of Minimax Trees and Game Searching Procedures", Artificial Intelligence 14 (1980), 113-138.

SLAG69 J.R. Slagle and J.K. Dixon, "Experiments with some Programs which Search Game Trees", JACM, Vol. 16, No. 2 (1969), 189-207.

ZOBR70 A.L. Zobrist, "A Hashing Method with Applications for Game Playing", TR 88, Computer Science Dept., Univ. of Wisconsin-Madison, 1970.

Number of Terminal Nodes Evaluated (5-ply)										
board	full window		no table		refutation table			transposition table		move
	direct	iterative	asp	PVS	full	asp	PVS	asp	PVS	
A	(forced	mate)								d6d1
B	61773	68399	46732	50625	69198	46485	48196	44052	46810	e4e5
C	50861	57539	34332	41019	34208	28227	30484	27300	30275	e8d8
D	58622	59437	55549	54294	50398	49370	48410	47226	47151	e5e6
E	180659	196349	94730	97074	111465	88807	88125	84515	84068	a1d1
F	24645	27364	20285	14151	26162	19472	14020	12579	12413	g5g6
G	116933	136416	84855	75801	94992	65194	60817	62586	57342	a3b4
H	7612	9116	8253	6124	5481	5108	4706	4086	4107	a2a3
I	132306	144505	86565	80933	81554	66957	67822	62556	67150	a2a4
J	181883	192933	112237	104027	127312	80331	80974	84774	79273	f6d7
K	109371	119427	56635	62999	65390	52342	51954	48968	48772	g3f5
L	78580	82392	43260	53514	53708	38600	44420	35853	38661	d7f5
M	143048	152922	139816	92164	111316	107346	85779	89234	82629	a1c1
N	31812	31701	31418	29875	30573	30273	29834	29694	29664	d1d2
O	34092	27048	25084	23459	22788	22225	21550	21652	21528	g4g7
P	75841	56372	51801	42900	50007	48075	40102	40518	39647	g5e7
Q	85844	91284	72159	62378	51742	41842	37859	33933	33924	d7b8
R	188877	201361	188009	128565	142188	134292	97861	94138	87243	g7h8
S	65370	82351	47504	52128	71536	43645	43762	41197	41512	a6a5
T	264078	287118	224568	171356	97785	78942	74026	130266	92728	a2a4
U	257810	223869	152228	124901	138113	107773	96104	99303	94603	f5d4
V	54032	64938	51318	45695	49705	43818	41810	39644	39178	e7d8
W	142147	307806	275530	212299	222935	192615	186438	179855	159550	g7g6
X	68567	73174	68008	71768	69627	67835	67803	67514	67515	b4c5
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.

Terminal node count and VAX/Unix@ CPU time (6-ply)							
	full window		transposition table		transposition and refutation table		move
	direct	minutes	PVS	minutes	PVS	minutes	
A	(forced	mate)					d6d1
B	157843	44	118055	40	92776	37	e4e5
C	270258	110	578855	190	130859	52	e8d8
D	100498	23	151945	30	96232	15	e5e6
E	502855	181	367191	122	347057	112	a1d1
F	48980	5	30675	3	27743	2	g5g6
G	552347	251	499806	220	410734	207	h5f6
H	26314	2	10985	1	9236	1	a2a3
I	547563	456	316397	193	272255	174	c3b5
J	606872	206	579776	192	221923	83	d8d5
K	303384	107	193808	67	166732	58	g3f5
L	414277	82	283386	68	138463	31	d7f5
M	299146	96	275861	75	201496	55	a1c1
N	87188	13	73899	11	68442	10	d1e1
O	123317	25	44297	14	40912	12	g4g7
P	172337	60	151980	39	150085	38	d2e4
Q	519506	307	228934	163	173727	115	d7b8
R	833502	424	548380	240	362541	155	g7h8
S	366195	82	256142	68	201459	51	a6a5
T	1286679	435	695456	241	664082	235	c3b5
U	1019468	696	619352	378	327153	163	f5h6
V	237350	139	179015	76	269214	147	e7d8
W	1644898	421	3652276	1091	1268625	413	c8f5
X	106773	27	161748	35	161587	34	b4c5
Total	10227550	4194	10018219	3557	5803333	2201	
Mean	444676	182	435574	155	252318	96	
%	100	100	98	85	57	50	

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