A QUANTITATIVE STUDY OF REFINEMENTS TO THE ALPHA-BETA ALGORITHM

T.A. Marsland

Technical Report TR82-6

August 1982

A QUANTITATIVE STUDY OF REFINEMENTS TO THE ALPHA-BETA ALGORITHM

T.A. Marsland

Computing Science Department
University of Alberta
EDMONTON

08-31-82

Draft: McGill Cognitive Science Workshop

ABSTRACT

In a recent paper several refinements to the alpha-beta algorithm were described and some indication of their relative efficiency given. The earlier data on the effectiveness of these enhancements is scattered throughout the literature and is not always directly comparable. To provide more consistency the results of a new study are presented. Rather than relying on searches of specially constructed trees, a simple working chess program was used to produce the data.

ACKNOWLEDGEMENT

Summer assistant Tim Breitkreutz implemented various alpha-beta refinements, and developed data reduction techniques to produce the tabular data from the chess program output. His enthusiasm and careful work were much appreciated.

Technical Report

TR82-6

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 nodes correspond to positions in the game tree and branches to moves). The leaves on the tree are called terminal nodes, and the others are referred to as interior nodes. To determine the best move in a zero sum game it is necessary to perform a minimax search of the whole tree. It is assumed that there are two players, here called Max and Min, who compete against each other. Basically, Max selects a move which maximizes his winnings, under the assumption that Min will choose responses that minimize them. For some games, like Chess, an exhaustive search of this tree is not possible, and so the outcome of the game is approximated by searches on trees of some fixed length. At a terminal node an evaluation function is used to estimate the value of the subtrees discarded at that point. The evaluation function itself may be very simple, computing only material difference, but must ensure that all terminal nodes are quiescent. In the case of Chess programs, this is done by building at the terminal nodes search trees which contain only check or capture moves. This subset search proceeds until either the position is quiescent (there are no more checks/captures) or some maximum depth of search is reached.

The alpha-beta algorithm achieves the same result as minimax, but does so more efficiently by employing two bounds which form a window. If this window covers the full range of values that the evaluation function can produce, then a full window search is being done. A call to the alpha-beta function could be of the form:

V = AB(p, alpha, beta, depth);

where p is a pointer to a position state vector, 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 value of the tree, and measures the potential success of the player to move. A skeleton for this function, expressed in the C language with Pascal style declarations and loops, appears in Figure A1 of Appendix A. The algorithm is expressed in a negamax framework [KNUT75], and so avoids the need for alternate min/max operations by always returning the negative of the subtree value from node to node. Undefined are functions evaluate(), to assess the value of the terminal nodes, generate(), to list the moves for the current position, make(), to actually play the move under consideration and undo(), to retract the current move.

In a recent survey paper [MARS82] several refinements to the alpha-beta algorithm were described and some indication of their relative efficiency given, based on the stated results of other authors. The previous data, however, is scattered and not always directly comparable. To provide more consistency the results of a new quatitative study are presented in Tables 1, 2 and 3. The results were produced by a simple working chess program[1], and these may be compared with those from searches of specially constructed trees [CAMP83].

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.

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. To determine exactly how much a shallow search may improve a deeper one was the aim of this study. The methods considered were:

(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 within which the score for the next iteration is expected to fall. In our study the width of this window is equal to two times the value of the smallest piece (a Pawn). Narrower windows are possible and sometimes quite successful, but the best way to choose the window size is not yet known. In any case 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 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 is shown in Appendix A, Figure A2.

(c). Minimal window search, in which the assumption made is that the first move to be tried is, with high probability, 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. This method, originally referred to as palphabeta but renamed Calphabeta [FISH81], will now be called principal variation search or PVS for short, in order to avoid confusion with parallel implementations [FINK82]. It is more or less equivalent to SCOUT [PEAR81][CAMP83]. The form of PVS is shown in Appendix A, Figure A3.

Both aspiration and minimal window searches can benefit further from the use of various tables. This study includes the use of refutation and transposition tables.

3. MEMORY TABLES.

Installation of a refutation table is straightforward and has low space overhead. After a search of depth D on a tree of constant width W the table will contain W*D entries. For each variation the table contains the sequence of D moves which determined a sufficient value for that variation. Thus for Figure 1, a tree of constant width W = 3 and fixed depth D = 3, the refutation table will contain the three sequences of three moves corresponding to the solid branches. In Figure 1 the branches with solid and double dotted lines are the ones actually searched, while those marked with single dots were cut off by the alpha-beta algorithm, i.e., were not examined at all. The numbers at the terminal nodes were produced by the 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. In this example the value of the tree is 4.

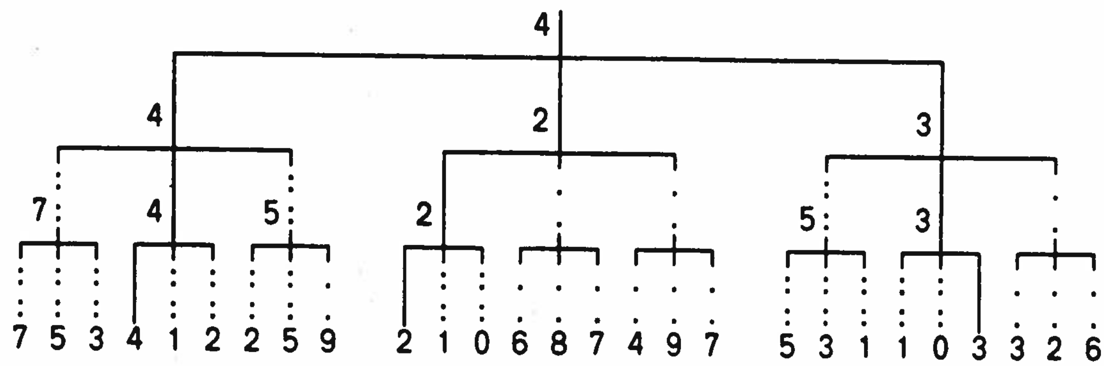

Figure 1: A 3-ply minimax tree showing refutation entries.

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 as normal. The candidate principal variation is fully searched, but for the alternate variations the moves in the refutation table may be sufficient to again cut off the search and thus save the move generation that would normally occur at each node. If the maximum length of the refutation path is 5 and the maximum tree width is 100 then, if each entry needs 2 bytes, just 1000 bytes are required to hold all the refutation lines for the current position. A small triangular table is also needed to identify the refutations [AKL77].

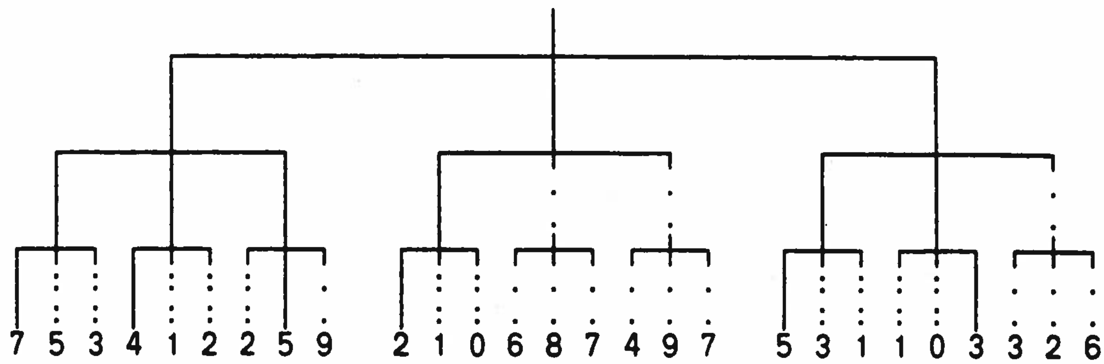

Figure 2: 3-ply tree showing transposition table entries.

A <u>transposition table</u> may also be used to hold refutations but, because it has the capacity for including more information, it has other capabilities too. In Figure 2 the positions actuall stored in the table are again shown by the solid lines. From thi we can see that the results from 15 positions would be stored, rather than only 9 in the refutation table case. Thus the 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 ar length of the subtree emanating from that point, then the transposition table may be used to extend the effective search depth [MARS82]. 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 move at the first level will be tried in the best possible order. A typical transposition table might contain 10,000 entries, each c 10 bytes [MARS82], for a 100,000 byte total storage overhead.

4. HYPOTHESES.

Based on a general understanding of the alpha-beta algorithm, one would expect that an iterative search will be better than a direct search, because successive refinement of the principal variation allows for increasingly large cut offs in the subtrees of the alternate variations. Simple iteration may not offer much improvement because only a single move is being used to seed the next iteration, and thus no additional cut offs will occur in the candidate principal variation where the largest subtree generally exists. With the aspiration searches, use of a narrow window has great potential for reducing the search time. In essence, the expected value of the principal variation is becoming known with greater certainty, and so the gamble that the true principal variation will remain within the window is less likely to be wrong. Thus aspiration searching should, on average, be significantly better than full window searching. Use of a refutation table should provide additional improvement, since the search is being guided directly to previous refutations. Finally, with a direct access transposition table further improvement should be possible. Just how important this may be is hard to predict, since it depends on what potential there is for improvement. Also, one major benefit of a transposition table (direct use of results from a previous subtree search) does not come into play until the depth of search is greater than 4. Since transposition table management is relatively complex, we can reasonably expect the efficiency of the method to be implementation dependent.

5. 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, our test program, and chess programs in general, perform significantly more calculation at a terminal node than at interior nodes in the tree. One reason for this is that a check or capture analysis in the form of an extended tree search is done. Therefore the following comparison will be based on the number of terminal nodes examined, especially since it has the additional advantage of being a machine independent measure.

6. 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, which are displayed in Appendix B for easy reference. The first position (Board A), however, is a simple problem of forcing checks and so was deleted from this study. 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 for the best method. The raw results are presented in Table 1, 2 and 3 respectively, along with the actual move made by the program in each case. For consistency the move selected should be independent of the algorithm, though it may change with depth of search, but this can only be guaranteed if the tree value for the best move is unique. In Tables 1, 2 and 3 the moves listed are from an iterative PVS search, while in Table 4 the moves produced by a direct full window search are shown. No matter which algorithm was used, the program correctly rated 6 of the 24 positions, in both the 4 and 5-ply cases, giving it an estimated USCF rating of 1550 [BRAT82], while for the 6-ply search 9 positions were solved (see Table 6).

Because the number of terminal nodes is exponential with the depth of search, the average terminal node count is plotted on a log-linear graph, Figure 3. 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 4, which shows the ratio of the terminal nodes searched relative to a direct search. From Figure 3, one may also deduce that for our data the incremental cost of a 4-ply search after a 3-ply one is a factor of 4.2. On the other hand the factor from 4 to 5 ply is 8.6, but from 5 to 6 ply the factor falls back to 5. Earlier experimental results [GILL72] suggested a factor of 7 for the addition of an odd ply and a factor of 3.5 for an even ply. While our results do not match Gillogly's exactly they are similar in form, and correspond quite well with some earlier ones [SLAG69] which noted factors of 8 and 4 respectively.

In order to provide a lower bound on the number of terminal nodes for our choosen data set it is necessary to estimate the 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, Table 4. On trees of constant width W and fixed depth D, there is a

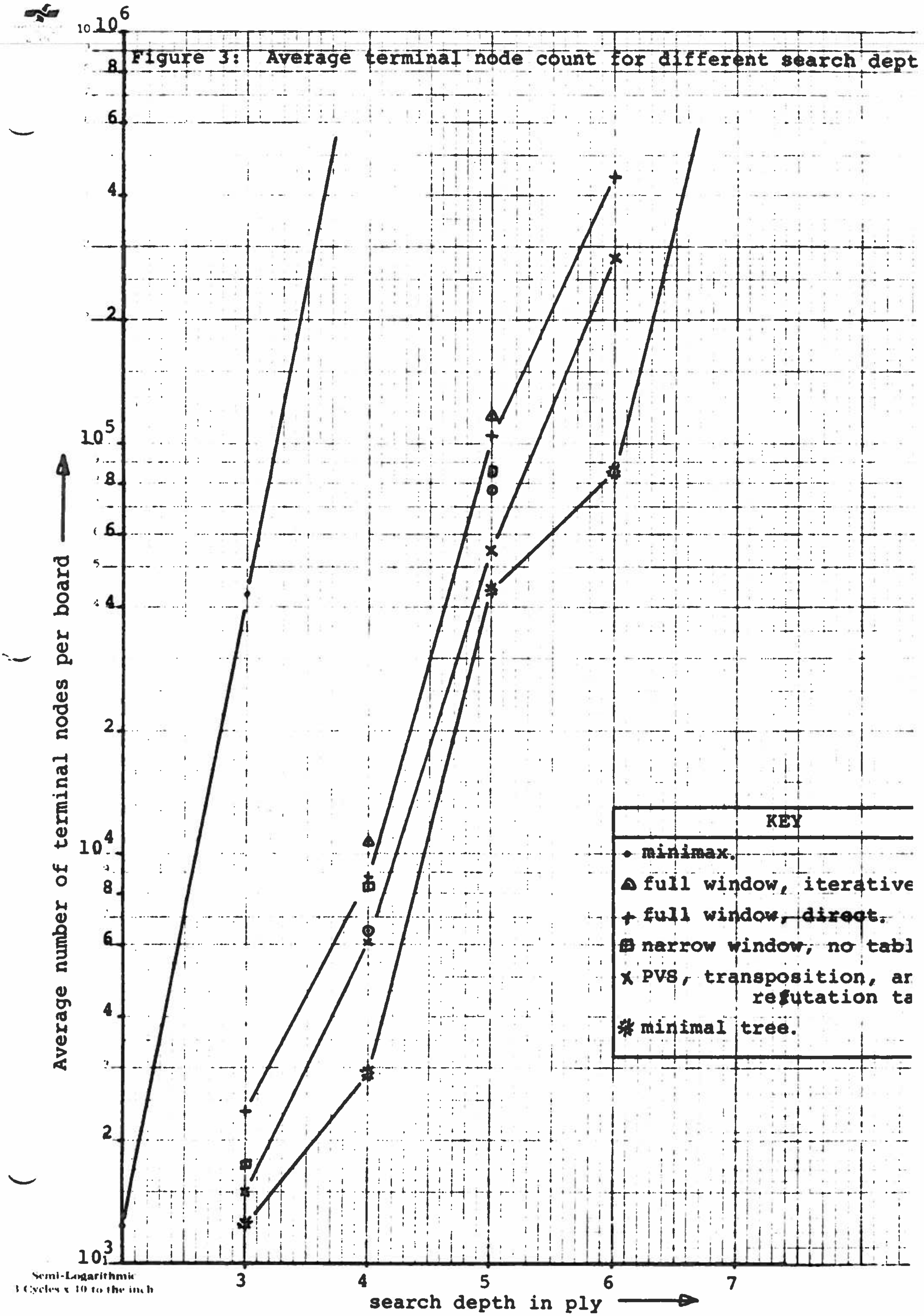

Figure 3: Average terminal node count for different search dept

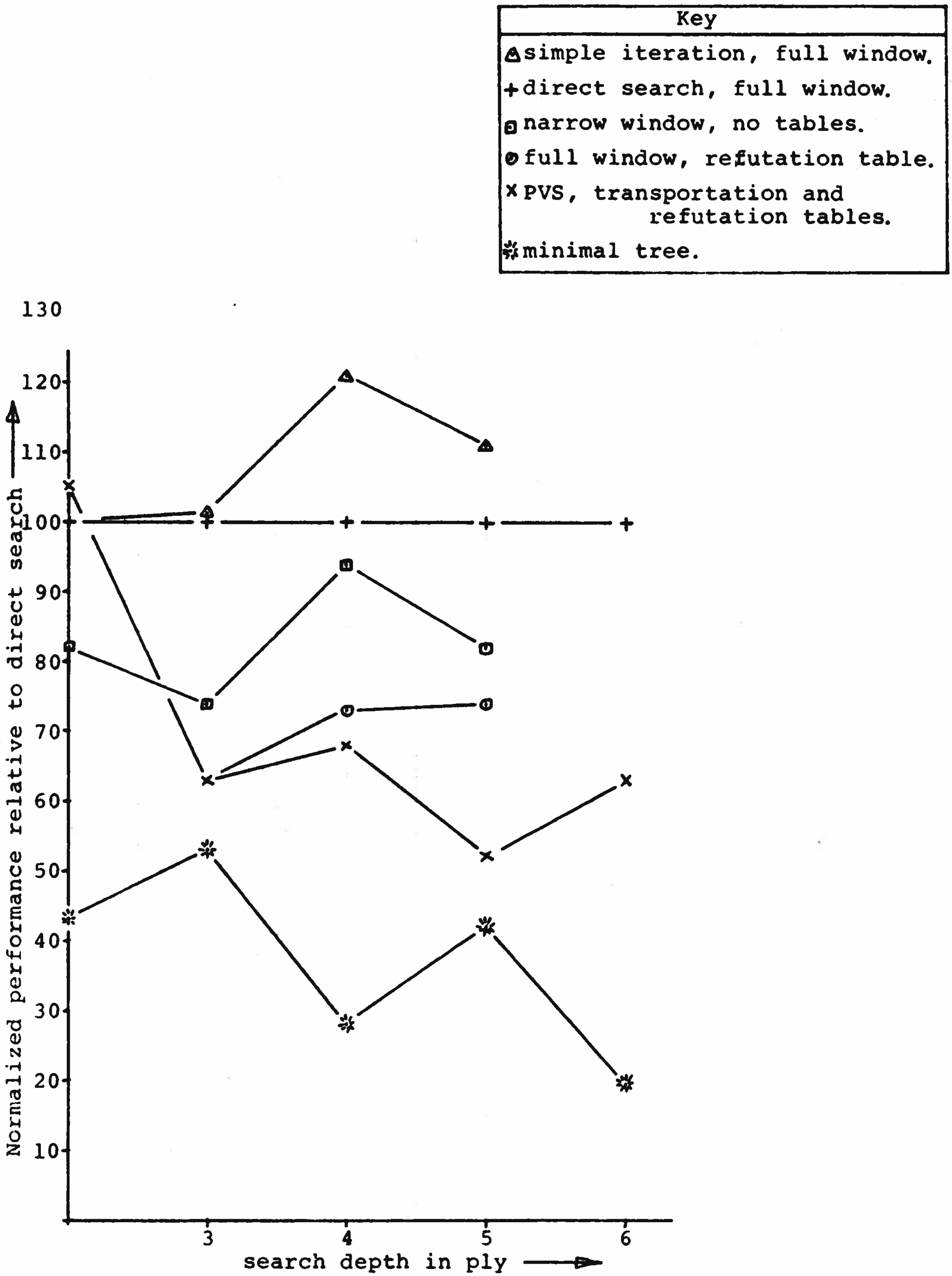

Figure 4: Performance comparison of alphabeta enhancements

formula for the minimal size of the tree that must be searched the alpha-beta algorithm, 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 conditior in Figures 3 & 4, and one can see that a factor of 1.3 reductic is possible on 3 and 5-ply trees and a factor of about 3.0 on 4 and 6-ply trees. The true reason for this difference is not clear. Possibly the data set of 23 positions is too small or is biased in some way. From Table 2 we see that boards T, U and W warrant particular attention, since they produced the largest trees. In the case of board W a change occurred in the principa variation, thus the 4-ply search was not a good predictor of th 5-ply result. Just how serious this can be is clear from Table which shows that for board W all the iterative searches are at least a factor of 2 more expensive than a direct search. This reinforced in the 6-ply results, Table 6, when for the PVS case the choosen move for board W again changes and 28% of the effor is expended on this one position.

In order to determine whether terminal node count is an adequate measure of an algorithm's performance, Table 5 is presented. Here the 5-ply results for the various refinements compared on a CPU utilization basis. For reasons already given the cost of generating an internal node is negligible in comparison to the cost of a terminal node. Although the cost o terminal node is quite variable, since a selective capture/che search is usually done there, simply counting the number of terminal nodes allows an adequate estimate of the efficiency o the algorithms to be made. From both Tables 3 and 5 the most efficient algorithms may be identified. 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 3).

7. CONCLUSIONS.

These results confirm our hypothesis that iterative deepening is effective if it is used in conjunction with some form of aspiration search. Further improvement is possible through the use of memory tables. Although the transposition table results were consistently better for the deeper searches they showed only marginal advantage over the cases using a refutation table. There may be a number of explanations for this. For example, since the positions were mostly from the middle game in chess, there were fewer possibilities for true transposition of positions. Also, for technical reasons, the best move at a terminal node was not recorded in the table. This is normal in the transposition case since it saves filling the table with large numbers of positions whose result is based on a very shallow tree. A final possibility is that our transposition table implementation was not the best.

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, 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 is unacceptably slow. For example, in the BLITZ[2] chess program a secondary sequence length of 10 is used, while BELLE[3] only the initial entry is considered. This latter appro: is simpler and was adopted in this study, even though our transposition table of 8192 entries was comparatively small. O data suggests that actual usage of this transposition table wa: far from perfect, and so further improvements may be possible. Determining the most effective way of using a transposition ta is very important, since it is clear from Figure 3 that there still considerable scope for improvement in these algorithms, especially in the even ply cases.

For the relatively shallow trees considered here there wa not much to choose between refutation and transposition table use. 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 guarantee that all the primary refutations will be retained. Since the refutation table is small and easy to maintain it is recommended that it always be updated and used whenever the transposition table fails to provide a primary refutation. In experience, the combination memory function is never significantly worse than use of a transposition table alone. F the 5-ply PVS case, we observed a 2 percentage point improveme on average while the overhead was negligible. In the 6-ply ca: more dramatic 22 percentage point improvement was seen (Table On the other hand, the true power of a transposition table wa:

2: BLITZ, a master calibre chess program developed by R. Hyat Univ. of Southern Mississippi.
3: BELLE, the current world champion chess program, developed K. Thompson, Bell Laboratories.

not brought out in this study, since there were few endgame positions.

Based on these experiments 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 [MARS82] in the efficiency of tree search algorithms.

Of the two principal refinements: narrow or minimal window aspiration search and memory tables, it is clear that preservation and use of the refutations from a previous iteration is more important than aspiration searching. This can be seen most clearly from Tables 1, 2 and 3, where full window aspiration searching supported by a refutation table is more powerful than aspiration searching without the help of a memory table. These two effects are not additive, but when combined a further improvement in performance occurs, especially for the deeper searches.

REFERENCES

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

BRAT82 I. Bratko and D. Kopec, "A Test for Comparison of Hum: and Computer Performance in Chess", Advances in Computer Chess 3, M.R.B. Clarke (editor), Pergamon Press, 1982.

CAMP83 M.S. Campbell and T.A. Marsland, "A Comparison of Minimax Tree Search Algorithms", Artificial Intelligence, (to appear) 1983.

FINK82 R.A. Finkel and J.P. Fishburn, "Improved Speedup Boun for Parallel Alpha-beta Search", Artificial Intelligence (to appear) 1982.

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.

MARS82 T.A. Marsland and M. Campbell, "Parallel Search of Strongly Ordered Game Trees", Computing Surveys (to appear) 1982.

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

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

Number of Terminal Nodes Evaluated (3-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	1995	2088	1260	1321	1234	1211	1239	1211	1239	f2g2
C	1356	1406	942	1004	901	901	905	901	905	h7h5
D	1758	1706	1693	1720	1644	1636	1670	1636	1670	d4b5
E	5501	5659	2815	2906	2674	2674	2676	2680	2682	a1d1
F	533	570	570	571	570	570	571	570	571	d7d2
G	2747	3033	2037	2049	1700	1685	1665	1721	1667	a3b4
H	412	467	423	327	253	241	262	241	262	a2a3
I	2738	2899	1852	1739	1604	1635	1604	1635	1604	f3e5
J	4067	4355	2307	2409	1929	1749	2035	1749	2066	f6h7
K	3099	3228	1617	1705	1509	1509	1513	1509	1513	g3f5
L	1718	1727	1230	1443	1288	1169	1342	1169	1342	d7f5
M	3255	3360	3156	2074	1981	1981	1982	2067	1984	e4e5
N	1144	1183	1048	1155	1084	1043	1124	1043	1124	d1e1
O	1436	1425	1243	1156	1068	961	1113	961	1113	f1f6
P	1397	1371	1251	1343	1273	1196	1279	1196	1279	g5e3
Q	2074	2153	1627	1492	1089	1051	1096	1051	1096	d7b8
R	3194	3419	3298	2352	2072	2039	2077	2039	2077	g7h8
S	1736	1987	1283	1421	1344	1234	1372	1234	1372	d7g4
T	3371	3538	2499	2067	1812	1781	1821	1781	1821	d1d2
U	4724	4449	3071	2953	2876	2849	2748	2853	2752	f5d4
V	1431	1601	1379	1301	1259	1222	1260	1222	1260	e7d8
W	3337	1754	2123	1847	1651	1651	1593	1651	1593	e8g8
X	1684	1754	1714	1757	1714	1714	1715	1714	1715	b4c5
Total	54707	55132	40438	38112	34529	33702	34662	33834	34707	
Mean	2378	2397	1758	1657	1501	1465	1507	1471	1509	
%	100	101	74	70	63	62	63	62	63	

Table 1: 3-ply terminal node count for alpha-beta variations.

Number of Terminal Nodes Evaluated (4-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	5252	7223	4899	5044	4520	3905	3934	3876	3905	f2g2
C	7246	6744	5338	5570	3892	3868	3874	3868	3874	h7h5
D	2628	5544	5078	6091	5198	4858	5072	4720	4848	e5e6
E	10031	15690	8455	8686	7822	6619	6621	6663	6665	a1d1
F	1670	2484	2484	2509	2484	2484	2509	2484	2509	g5g6
G	14282	19805	17140	16624	10632	10409	9805	10798	10053	a3b4
H	1331	1695	1583	1343	907	862	869	845	866	a2a3
I	8657	11556	8582	7718	5961	5925	5426	5925	5426	f3e5
J	14013	16818	10943	10014	8598	6624	9434	6865	9739	d8d7
K	6828	10056	6314	6660	4984	4485	4467	4372	4379	g3f5
L	9289	9929	6067	10153	7622	5099	8898	5004	7977	d7f5
M	6514	9874	9244	6712	5854	5709	5711	6035	5792	e4e5
N	3005	3182	3588	3534	3050	3437	3505	3396	3497	d1d2
O	5093	4278	4584	3560	2662	3006	2654	3004	2652	g4g7
P	6517	4030	3738	3895	3713	3587	3620	3585	3618	g5e7
Q	12296	11676	9282	8739	6938	5630	5603	6232	6105	d7b8
R	11994	15195	13850	11447	9836	8710	8750	8858	8807	g7h8
S	8058	11572	7536	8006	8506	6026	6166	6118	6258	a6a5
T	21993	24707	21201	16790	14731	13920	13729	14234	13680	a2a4
U	19710	19214	14086	13634	9354	9177	8927	9562	9360	f5d4
V	5936	7571	5872	5484	6090	4938	4874	4794	4769	e7d8
W	18901	22724	18661	16518	11648	11437	10569	10515	10456	a8g8
X	2853	4607	4391	4514	4526	4387	4355	4348	4349	b4c5
Total	204097	246174	192916	183245	149528	135102	139372	136101	139584	
Mean	8873	10703	8387	7967	6501	5874	6059	5917	6068	
%	100	121	94	90	73	66	68	67	68	

Table 2: 4-ply terminal node count for alpha-beta variations.

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 3: 5-ply terminal node count for alpha-beta variations.

Average Width, Moves and Mean Time in mins.						
board	3-ply width	3-ply move	4-ply width	4-ply move	5-ply width	5-ply move
B	31	c1h1	26	c1h1	30	e4e5
C	30	e7d8	38	e7d8	31	e7d8
D	34	d4b5	27	e5e6	33	e5e6
E	45	a1d1	40	a1d1	44	a1d1
F	21	d7d1	17	g5g6	20	g5g6
G	33	a3b4	31	f3g3	35	f3g3
H	16	e2c3	17	e2c3	16	e2c3
I	38	f3e5	32	f3e5	38	f1d3
J	40	d8d5	37	d8d5	40	d8d5
K	36	g3f5	33	g3f5	35	g3f5
L	40	d7f5	41	d7f5	40	d7f5
M	36	e4e5	24	e4e5	34	a1c1
N	36	d1d2	41	d1d2	37	d1d2
O	40	f1f6	39	g4g7	40	g4g7
P	36	g5e3	32	g5f4	36	g5f4
Q	30	d7c5	39	d7c5	31	d7c5
R	42	c8g4	37	c8g4	40	c8g4
S	37	c7c5	43	c7c5	36	c7c5
T	38	d1d2	32	d1d2	37	d1d2
U	43	f5d4	31	f5d4	41	f5d4
V	37	d7e5	41	d7e5	38	d7e5
W	37	e8g8	42	c8f5	35	c8f5
X	37	b4c5	35	b4c5	38	b4c5
ave.	35		34		35	
time		0.9		4.5		33

Table 4: Average branching factors for position data.

Processor Time in VAX/Unix® minutes (5-ply search)										
board	full window		no table		refutation table			transposition table		move
	direct	iterative	narrow	PVS	full	narrow	PVS	narrow	PVS	
A	(forced	mate)								d6d1
B	18	20	16	18	26	19	19	18	19	e4e5
C	25	33	17	22	18	14	17	13	17	e8d8
D	11	11	10	9	10	9	8	8	8	e5e6
E	40	48	27	27	33	24	24	22	22	a1d1
F	2	2	1	1	2	1	1	1	1	g5g6
G	43	54	36	30	42	31	29	29	25	a3b4
H	0	1	1	0	0	0	0	0	0	a2a3
I	49	60	33	29	30	24	23	22	22	a2a4
J	60	68	52	48	46	31	30	34	30	f6d7
K	32	36	19	19	23	16	16	15	14	g3f5
L	19	23	12	18	17	11	16	10	12	d7f5
M	20	22	20	14	17	16	13	13	13	a1c1
N	6	6	5	5	5	5	5	5	5	d1d2
O	7	7	6	5	8	8	7	8	6	g4g7
P	16	12	11	8	11	10	8	8	8	g5e7
Q	75	80	75	59	45	40	32	31	30	d7b8
R	63	68	66	43	53	50	32	30	27	g7h8
S	15	24	14	15	22	14	14	13	13	a6a5
T	63	70	57	44	30	25	24	36	27	a2a4
U	98	93	71	48	48	41	31	37	31	f5d4
V	22	33	21	17	23	16	14	14	13	e7d8
W	52	104	162	136	138	125	130	116	114	g7g6
X	14	19	16	16	16	15	15	15	15	b4c5
Total	752	892	746	631	664	546	509	499	471	
Mean	33	39	32	27	29	24	22	22	20	
%	100	119	99	84	88	73	68	66	63	

Table 5: Processor time in minutes 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	122232	43	111614	44	e4e5
C	270258	110	282301	108	234145	100	e8d8
D	100498	23	100473	17	97686	16	e5e6
E	502855	181	374464	125	352833	117	a1d1
F	48980	5	35842	3	38868	3	g5g6
G	552347	251	501514	209	423719	190	h5f6
H	26314	2	13510	1	11949	1	a2a3
I	547563	456	349796	219	298526	196	c3b5
J	606872	206	428571	148	237638	89	d8d5
K	303384	107	212617	78	171385	61	g3f5
L	414277	82	324169	71	265629	59	d7f5
M	299146	96	300040	81	277068	76	a1c1
N	87188	13	80357	12	79384	12	d1e1
O	123317	25	57373	14	56578	15	g4g7
P	172337	60	198756	63	201648	62	d2e4
Q	519506	307	298827	194	148777	108	d7b8
R	833502	424	571416	242	407650	168	g7h8
S	366195	82	280452	73	226871	64	a6a5
T	1286679	435	763514	265	292138	101	c3b5
U	1019468	696	573348	353	347795	153	f5h6
V	237350	139	165344	76	147418	59	e7d8
W	1644898	421	2459242	671	1808486	515	c8f5
X	106773	27	162157	36	161829	34	b4c5
Total	10227550	4194	8656315	3103	6399634	2243	
Mean	444676	182	376361	135	278244	98	
%	100	100	85	74	63	54	

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

```
function AB(p : position; alpha, beta, depth : int) : int;
{
  VAR width, score, i, value : int;

  if (depth ≤ 0)                        /* a terminal node? */
     return(evaluate(p));
                                   /* determine successor positions */
  width = generate(p);             /* p.1 ... p.w and return number */
                                   /* of moves as function value    */
  if (width == 0)                        /* no legal moves? */
     return(evaluate(p));
  score = -INF;
  for i = 1 to width do {
     make(p.i);
     value = -AB(p.i, -beta, -max(score,alpha), depth-1);
     undo(p.i);

     if (value > score)                /* an improvement? */
        score = value;
     if (score ≥ beta)                 /* a cutoff? */
        return(score);
  }
  return(score);
}
```

Figure A1: Depth-limited alpha-beta function.

```
/*    Assume V = estimated value of position p, and
             e = expected error limit.
*/
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 A2: Iterative deepening with aspiration search.

```
function PVS( p : position; depth : int) : int;
{
  VAR width, score, i, value : int;
  if (depth ≤ 0)
     return(evaluate(p));
  width = generate(p);
  if (width == 0)
     return(evaluate(p));
  make(p.1);
  score = -PVS(p.1, depth-1);
  undo(p.1);
  for i = 2 to width do {
     make(p.i);
     value = -AB(p.i, -score-1, -score, depth-1);
     if (value > score)
        score = -AB(p.i, -INF, -value, depth-1);
     undo(p.i);
  }
  return(score);
}
```

Figure A3: Minimal window search.

```
   Kb    Rb    ::    ::
Pb Pb :: Bb :: Rw ::
   ::    Qb    :: Pb Pb
::    ::    Pb    ::
   :: Bw ::    ::    ::
::    ::    QW    ::
Pw Pw Pw ::    Bw    ::
::    KW    ::    ::
JUL 31,       BLACK'S MOVE.
```

2K5PPP2B24Q32B5885R28-88884P33Q2PPPP1B41K1R4-
Board A: Best move is: Qd6d1+

2R55K21NR5P2PPPP17P+8888P71PPR3P4NPP13R1K2+
Board B: Best move is: d4d5

7R2BQ1PRK1P2BNNPP1P1P1P13P4-8881P62P1P3P2P1PP13BBNNP2Q1RR1K-
Board C: Best move is: f6f5

R1B1KB1RPPP1QPPP2P53N44P3+88882PP41P6P3PPPPRNBQKB1R+
Board D: Best move is: e5e6

2KR1B1RPPP3PP2N2N1Q3P1P1B8+88883P3P2N1PP2PBPQ42KR1BNR+
Board I: Best move is: f4f5

R1B3K12P1NQPPPP1R1P288-8883P481QN2NP1PP3PP13RR1K1-
Board J: Best move is: Nf6e5

R4RK14QPPP2PBB1N1P1P1P33P4+8888N1P1P3BP1P4P2Q1PPP2R1NRK1+
Board K: Best move is: f2f4

R3R1K1PP3PPP2P586NQ-88884P38PPQB1PPPR3R1K1-
Board L: Best move is: Bd7f5

R1B2RK1PPB1QP22N2N1P2P1P1P13P4-88884P3P2P1NB11PPNBPPPR2Q1RK1-
Board Q: Best move is: h7h5

R1B1QRK11PB3PPN1P2N2P3PP28-88882N52NP2P1PP2PPBPR1BQ1RK1-
Board R: Best move is: Ne5b3

4RRK1P5PP1P62QBPP28-88888P2P1PNP2PQ2PK3RR3-
Board S: Best move is: Re8xe4

2KRR3PPP1Q2P2N3P13P1P24N3+88883P1P21P1QP2PPB2BP1RR4K2+

```
Rb :: Bb ::    Rb Kb ::
::    Qb    Bb    Pb Pb
Pb ::    Pb Pb Nb    ::
:: Pb ::    ::    ::
   ::    QW Pw ::    ::
:: Bw Nw    Bw    ::
Pw Pw Pw ::    :: Pw Pw
Rw    ::    :: Rw KW
JUL 31,       WHITE'S MOVE.
```

R4RK1PPP3PP1BN1B33QP38+88881P6P2PPN22Q1B1PPR1B2RK1+
Board E: Best move is: Nc3d5 or a2a4

8P1P51P4K15P24P1P183R48+888884P3PPP2PP12R3K1+
Board F: Best move is: g5g6

R2Q2K12P1B1PPB1P2R23P1P24P2N+888B1P5Q2P1P24P3PP2N1PP1NK1R1R1+
Board G: Best move is: Nh5f6

8P3N2P4K1P13P1P24P3+8882P53P3P6P1P3KP24B3+
Board H: Best move is: f4f5

R3R1K1PP1B1PPP3Q44P33P4+8883P42P5P2P44BPPPR2Q1RK1+
Board M: Best move is: b2b4

R2Q1RK1PB2PPBP1P3NP1B3P4+882B58Q4P22PP2P1PP2P2PRNB2R1K+
Board N: Best move is: Qd1d2

5RK1P1PN2PP1P1PP1R16Q18+8888P1PP42B1PR21P2Q1PP2R3K1+
Board O: Best move is: Qg4xg7+

R2Q1RK1PPPN1PPP1B684P1B1+88881P1P4P1P4P4NPP1R1BQKB1R+
Board P: Best move is: Nd2e4

3RR1K1PB3PPP1P5Q2P1P35N2+888882PPQP2PPB2RPP3RN2K+
Board U: Best move is: Nf5h6

R2R2K11B2QPP1P2B1N1PN1P1P31P6-8888P71P1PPN1P1BQNBPP12R2RK1-
Board V: Best move is: Bb7xe4

R4RK11PP3PP2N1B3P2Q1P24P3-88883P42P5PP2BPPPR1BQK2R-
Board W: Best move is: f7f6

5RK13QNPPPPRNBB31PP1P33P4+88882P1PP21P1P2PPP2B2B1R2QNRNK+
Board X: Best move is: f2f4

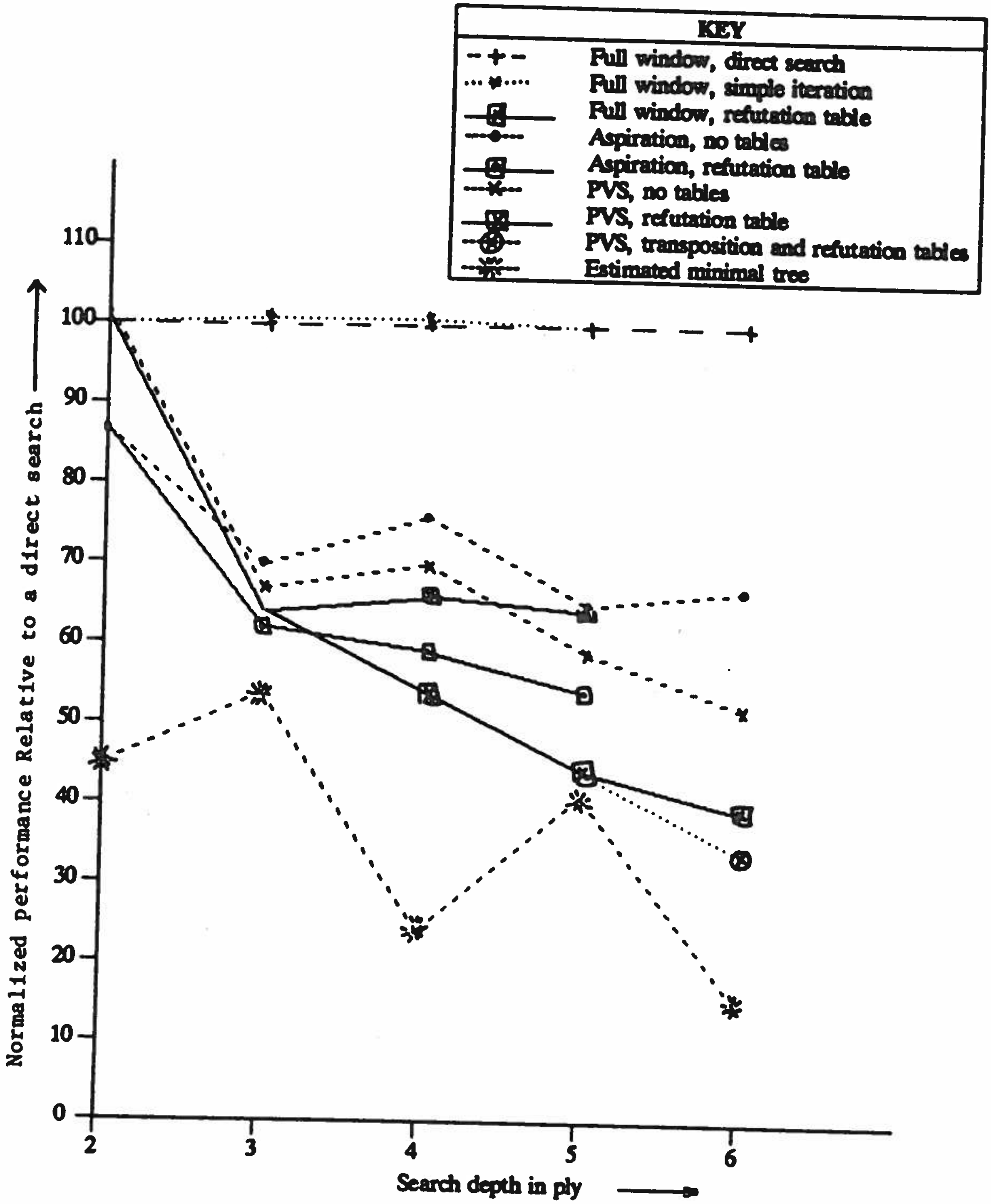

Figure 4: Performance Comparison of Alphabeta Enhancements