Replacement schemes for transposition tables

Breuker, D.M.; Uiterwijk, J.W.H.M.; van den Herik, H.J.

Published in:
ICCA Journal

Publication date:
1994

REPLACEMENT SCHEMES FOR TRANSPOSITION TABLES

D.M. Breuker, J.W.H.M. Uiterwijk and H.J. van den Herik[1]

Maastricht, The Netherlands

ABSTRACT

Almost every chess program makes use of a transposition table, typically implemented as a large hash table. Even though this table is usually made as large as possible, subject to memory constraints, collisions occur. Then a choice has to be made which position to retain or to replace in the table, using some replacement scheme. This article compares the performance of seven replacement schemes, as a function of transposition-table size, on some chess middle-game positions. A two-level table, using the number of nodes in the subtree searched as the deciding criterion, performed best and is provisionally recommended.

1. INTRODUCTION

Chess programs analyze positions while building *trees*. However, a closer look shows that the search space could better be explored by *graphs*, due to the fact that a position can be reached by several orderings of moves. Such resultant positions are known as *transpositions*.

When encountering a position again, the size of the search tree can be reduced considerably if the previous results for this position are still available. This information can be stored in a large direct-access table, the *transposition table* (Greenblatt *et al.*, 1967; Slate and Atkin, 1983). Relevant information for an entry in the transposition table includes the score of the position, the best move and the depth of the subtree searched. Since we adhere to α-β search (Knuth and Moore, 1975), the score need not be a true value, but may be a lower or upper bound. When using iterative deepening and minimal-window search, transposition tables may significantly reduce the search effort (Ebeling, 1986; Berliner and Ebeling, 1989; Schaeffer, 1989; Hyatt *et al.*, 1990), especially in endgame positions with few pieces on the board.

The literature on transposition tables is mainly tutorial in nature (for example, Marsland, 1986), with only a few detailed discussions of performance (for example, Ebeling, 1986; Schaeffer, 1989). One frequently cited performance observation is that doubling the size of the table reduces the size of the search tree. This is an obvious result, since the more the information in the table, the greater the chance of finding a transposition. Performance analyses of other aspects of transposition tables, such as which positions to replace, have not, as far as we know, been published in the literature.

The most common implementation of a transposition table is a large hash table. Even though this table is usually made as large as possible, subject to memory constraints, collisions (for which see Section 3) are bound to occur. When a collision occurs, a choice has to be made whether to replace or to retain the position in the table. This choice is governed by a *replacement scheme*. From the literature and from discussions with computer-chess practitioners, it appears that the most common form of collision resolution is to prefer the results of deeper searches over shallower ones. This has an intuitive appeal, but has not been supported experimentally.

---

[1] University of Limburg, Department of Computer Science, P.O. Box 616, 6200 MD Maastricht, The Netherlands. Email: breuker@cs.rulimburg.nl, uiterwijk@cs.rulimburg.nl, herik@cs.rulimburg.nl.

This article compares the performance of seven collision-resolution schemes on eighteen chess middle-game positions. Perhaps surprisingly, we find that the traditional implementation (one table position per entry) and the preference for retaining the results of deep searches over shallow ones is not the best. We adopt a scheme first proposed by Ebeling (1986): a two-level table. We examined both the depth of the subtree searched and the number of nodes as the replacement criterion. In general, the latter performed best in the positions tested.

Details related to implementing transposition tables are described in Section 2. Section 3 identifies the problems that arise when using transposition tables. Relevant details of the implementation of the chess program used in our experiments are given in Section 4. In Section 5 our set of test positions is discussed. The design of the experiment is described in Section 6. The results of our experiments are presented and compared with results in the literature in Section 7. Finally, in Section 8 conclusions and proposals for future research are given.

2. IMPLEMENTING TRANSPOSITION TABLES

In the ideal case one would preserve every position encountered in a search process, together with all its relevant information. Unfortunately, the memory then required exceeds the available capacity of most present-day computers. Therefore, in practice, a transposition table is usually implemented as a *hash table* (Knuth, 1973). A position is converted to a *number* of potentially sufficiently large size (the *hash value*) by using some hashing method. The most popular method used by chess programmers is described in Zobrist (1970).

If the transposition table consists of 2^n entries, the n low-order bits of the hash value are used as a *hash index*. The remaining bits (the *key*) are used to distinguish among different board positions mapping onto the same hash index (i.e., the same entry in the transposition table). Therefore, the total number of bits should be chosen sufficiently large.

Transposition tables are usually implemented as one table with a fixed number (traditionally one) of table positions per entry (Slate and Atkin, 1983). Sometimes an overflow area is used for handling a limited number of collisions.

An entry in a transposition table should at least contain the following information (Marsland, 1986; Hyatt *et al.*, 1990):

Key[1]: The bits of the hash value more significant than those of the hash index. The key is needed to distinguish among different board positions having the same hash index.

Move: The best move in the position. This is the move which either caused a cutoff, or obtained the highest score.

Score: The score of the best move in the position. Since we adhere to α-β search, the score can be a true value, an upper bound or a lower bound.

Flag: A field containing information concerning the score, indicating whether it is a true value, an upper bound or a lower bound.

Depth: The relative depth in the subtree searched. When doing an n-ply search and a position is stored at ply m of the tree, the depth is n-m.

---

[1] Marsland (1986) uses the term 'lock'.

3. ERRORS FROM TRANSPOSITION TABLES

Implementing a transposition table as a hash table introduces two types of error, identified as early as 1970 by Zobrist. The first type of error (called *type 1* by Zobrist (1970)) is the more important. Since the number of available hash values is much less than the total number of positions in chess, it must happen that two different positions yield the same hash value. This is a serious error, because when a type-1 error occurs, the information in this entry will be used in the wrong position and, if so, will introduce search errors. Often it is possible to detect this error by testing the move from the transposition table for legality in the position, effectively lowering the error rate. If the move is illegal, then the table entry must concern another position than the one being investigated. Note that if the move *is* legal, the positions still may differ. The probability of the occurrence of type-1 errors can be lowered by increasing the number of bits in the hash value.

The second type of error (called *type 2*, or *clash* by Zobrist (1970)), occurs when two different board positions map onto the same entry in the transposition table, i.e., the same hash index. This is commonly known as a *collision* (Knuth, 1973). When a collision occurs, a choice has to be made which of the two positions involved should be preserved in the transposition table. Such a choice is based on a *replacement scheme*. Several replacement schemes are discussed in Subsection 6.1. The probability of the occurrence of collisions can be lowered by increasing the number of bits in the hash index (i.e., the number of entries in the transposition table).

4. THE STRUCTURE OF THE IMPLEMENTATION

The test program ALIBABA is a simple chess program, designed to be easily reproducible by other researchers[1]. This reproducibility serves to promote a uniform platform for research. The major components of ALIBABA constitute the remainder of this Section, viz. the search engine (4.1), the move-ordering heuristics (4.2), the evaluation function (4.3) and the implementation of the transposition table (4.4).

4.1 The Search Engine

ALIBABA searches game trees using the α-β algorithm (Knuth and Moore, 1975). The likelihood of desirable cutoffs is increased when the best move is examined first at interior nodes. Therefore, *iterative deepening* (Scott, 1969) is applied, amounting to a full-width search to increasing depths. Information gathered from previous iterations can be used to re-order moves at interior nodes in the current iteration, increasing the likelihood of putting the best move at the front of the current move list.

Another α-β enhancement implemented is the *minimal window*, based on the fact that it is relatively easy to show that a move is worse than the best move found so far. The resultant algorithm is known as iterative-deepening, minimal-window, principal-variation search (PVS; for details, see Marsland (1986)).

Furthermore, ALIBABA uses *aspiration search* (for a description, see Marsland (1986)). At the start of each new iteration, the upper bound and lower bound of the window are set to the score resulting from the previous iteration *plus* and *minus* the value of a Pawn, respectively. If the search fails (the score does not lie within the α-β window), the window is adjusted to either $(-\infty, \text{score})$ when failing low, or $(\text{score}, +\infty)$ when failing high.

When evaluating nodes they should be "relatively quiescent" (Shannon, 1950). Not all leaf nodes are, so they must be investigated further by a *quiescence search*. In this search only capturing moves and promotion moves are considered, except when the King is in check, then all moves must be searched. We note that a quiescence search, when the player to move is not in check, may be terminated early, viz. as soon as it becomes clear that all moves to be generated will be disadvantageous.

---

[1] A full description of the program will be given in the forthcoming Ph.D. thesis of one of us (D.M.B.), who can also be contacted for a free copy of the full C source code, which is available for public distribution.

No other search extensions are used in our transposition-table experiments in order to avoid possible search anomalies.

4.2 The Move-Ordering Heuristics

In any position, ALIBABA generates only legal moves, excluding pseudo-legal moves, such as putting or leaving its own King in check. Since the ordering of moves is important for the efficiency of the α-β algorithm, the following ordering heuristics are implemented.

Refutation tables (Akl and Newborn, 1977). For every move in the root position, the principal variation is stored. In the next iteration, moves out of these principal variations are tried first.

History heuristic (Schaeffer, 1983, 1989). A score for every legal move encountered in the search tree is maintained. Every time a move is found to be best in a search, its score is adjusted by an amount proportional to the depth of the subtree investigated. When ordering moves with this heuristic, moves with a higher score are considered before moves with a lower score.

In ALIBABA, the moves are ordered in the following way. The first move to be considered is the move from the refutation table. Then, if the position is found in the transposition table, the transposition-table move is the next move to be considered. These moves are followed by capture moves (the highest-valued piece to be captured first; if equal, then the lowest-valued capturing piece first). Thereafter follow the promotion moves (ordered by promotion piece; the highest-valued promotion piece first). The remaining moves are ordered according to their descending history-heuristic scores. In addition to the move-ordering heuristics mentioned above, which are applied immediately after move generation, the root moves are also ordered during the iterative-deepening search processes.

4.3 The Evaluation Function

The evaluation function used for the transposition-table tests is simple. It consists of a material part and a positional one. The material part counts the difference in material between sides. The positional part is restricted to summing piece-square-table values. During a game, for every type of piece a 64-square table is maintained. Each table contains positional values for that piece on every square on the board. These piece-square tables are filled at the beginning of each new search process. The values, in turn, are derived from the mobility of pieces and the centre control. The technique of filling tables at the start of every new search process is called *preprocessing*. The positional part of the evaluation function is updated incrementally: whenever a move is investigated during the search process, the positional value of the piece-fromSquare table entry is subtracted from it, and the value of the piece-toSquare table entry is added to it. Finally, the evaluation function also serves to detect draws by stalemate, by three-fold repetition and by the 50-move rule as well as checkmate.

4.4 The Implementation of the Transposition Table

Whenever a move is investigated in the search the resulting position is looked up in the transposition table. If the position is present, and the depth of the examined subtree is greater than or equal to the depth still to be searched, the information in the table is considered reliable and therefore used to update the window bounds (possibly causing a cutoff). The transposition-table move is always used to order moves (see Subsection 4.2). In α-β search, after a position is investigated to a certain depth, it is stored in the transposition table together with the best move (i.e., the move which caused a cutoff, or the move with the highest score), its score, the search depth and a flag, denoting whether the score was a true value, a lower bound or an upper bound. During quiescence search, a position is never stored in the transposition table.

The results of a transposition-table lookup are used at *all* nodes in the tree. If a leaf position is present in the table, the transposition-table score is used for the evaluation. If the score was a true value, this score is returned. Otherwise, the position is evaluated and if necessary the evaluation value is adjusted according to the bound indicated by the flag in the transposition table. Since the evaluation function is used in the quiescence search, the transposition table is used in the quiescence search as well. Note, however, that, since entries are only retrieved and not stored during quiescence search, their usefulness is limited during that phase.

In ALIBABA, the transposition table is implemented as a linear array with one or two table positions per entry. No overflow area is used. More details of the implementation of a transposition table in plain α-β search are given in Marsland (1986).

5. THE TEST SET

For various reasons, available sets of test positions (Reinfeld, 1958; Kopec and Bratko, 1982; Nielsen, 1991; Lang and Smith, 1993) were unsatisfactory for our purpose. Instead, we have opted to use a sequence of positions derived from an actual game, so as to test a diversity of techniques.

One advantage of this is that the chosen positions will not be biased towards the tactical but will automatically incorporate the positional aspects. Moreover, this choice also meets the requirement that successive positions should be related, which is essential when investigating the effects of clearing the transposition table between moves.

Specifically, we have chosen positions of the game Kasparov-Short from round 2 of the Euwe memorial VSB tournament 1994 as our test set (see Appendix A)[1]. The opening phase is omitted for obvious reasons. We shall only consider positions from move 15 onwards. Also, we have concentrated on the use of transposition tables in the middle game to the exclusion of the endgame, the latter defined as a position in which at least one side has fewer than 18 points of material[2]. We note that the present test terminates when the game is still a middle game according to this definition. Our final restriction is that only Kasparov's positions are investigated[3], resulting in 18 positions as a test set.

6. THE DESIGN OF THE EXPERIMENT

In this Section, we report the testing of seven replacement schemes, the impact of clearing the transposition table between moves and the effect of changing the size of the table. We use the number of nodes investigated during a search as a measure, including *all* nodes, i.e., interior nodes and leaf nodes.

6.1 Replacement Schemes

Whenever a collision is detected, a choice has to be made whether to replace the existing position in the transposition table. In this Section we examine seven different *replacement schemes*, viz. DEEP, NEW, OLD, BIG1, BIGALL, TWODEEP, TWOBIG1. They are based on five concepts, as numbered below.

1. (DEEP)
The replacement scheme DEEP is traditional. It is based on the depths of the subtrees examined for the positions involved. At a collision, the position with the *deepest* subtree is preserved in the table (Marsland, 1986; Hyatt *et al.*, 1990). The concept behind this scheme is that a subtree searched to a greater depth usually contains more nodes than a subtree searched to a shallower depth. Therefore, more time was invested in searching the larger tree. Hence, storing this position in the transposition table potentially saves more work than storing a position less deeply investigated.

---

[1] We are using only one game in order to establish which of our experiments could be usefully run on a larger test set (e.g., all Kasparov games of the Euwe memorial VSB tournament 1994). Those results will be incorporated in the forthcoming Ph.D. thesis of the first author.

[2] Pawn=1, Knight=3.25, Bishop=3.25, Rook=5, Queen=9. Kings do not contribute.

[3] This could be interpreted as a bias in the test positions.

2. (NEW)
The replacement scheme NEW *always* replaces any position in the table when a collision occurs. This concept is based on the observation that most transpositions occur locally, within small subtrees of the global search tree (Ebeling, 1986).

3. (OLD)
We have also tested the replacement scheme OLD. With this scheme (the opposite of the scheme NEW) a new position *never* replaces an existing position. This scheme has only been included for the sake of completeness.

4. (BIG1, BIGALL)
Sometimes a subtree contains many forcing moves. It also may be potentially well-ordered (in which case many cutoffs have occurred). In such cases, the depth of the search tree fails to be a good indicator of the amount of search already performed and therefore potentially to be saved. It then may be attractive to select, for retention, the position with the *biggest* subtree rather than the one with the deepest subtree, going by number of nodes rather than by their depths. A drawback then is that the number of nodes must be retained as part of each transposition-table entry, reducing the effective number of entries possible for a given amount of storage.

This scheme can be implemented in two variations, say BIG1 and BIGALL. The former counts a table position in a transposition table as a single node, the latter as N nodes, where N is the number of board positions searched in order to obtain the information of the table position stored.

5. (TWODEEP, TWOBIG1)
Ebeling (1986) describes the use of a *two-level* transposition table. So does Schaeffer (1994), combining in the implementation the schemes DEEP and NEW. Such a transposition table has two table positions per entry[1]. Upon a collision:

- if the new position has been searched to a depth greater than or equal to the depth of the extant first-level-table position, the new position replaces the latter, whereas the extant position is shifted to the second-level position;
- otherwise, the new position is stored in the second-level position.

Thus, the newest position is always stored, and the less important of the remaining two positions (in terms of depth of search) is overwritten. We call this scheme TWODEEP. We also tested the analogous combination of the schemes NEW and BIG1 (further denoted as TWOBIG1).

We note that in all replacement schemes in our experiments the decision to overwrite an entry does not depend on the type of the score (true value, lower bound or upper bound) of the positions involved.

6.2 Time Stamping

When playing a game, a choice must be made what to do with the positions stored in the transposition table during the search from a previous position in the game. Successive positions in a chess game are related to one another, and it therefore may seem best to retain *all* positions in the transposition table. However, these positions are subject to aging, and will be of little use after a few moves in the game. Consequently, clearing the transposition table between moves may also seem attractive, e.g., when the evaluation function between searches is changed.

Instead of physically clearing entries in the transposition table, it may be preferable to time stamp them after the completion of each move. A time-stamped position remains stored in the table until a collision occurs, when it is unconditionally overwritten. While time stamped but not overwritten, it will still be used for retrieving information. A position not time stamped holds information more recent than any previous search.

---

[1] Ebeling (1986) implemented the two-level transposition table in a slightly different way.

6.3 Table Sizes

Undoubtedly, many experiments have been conducted to test the effect of the transposition-table size on the number of nodes investigated. In spite of this, there are few reports in the literature. Ebeling (1986) states: "each doubling in the hash table size yields only a 7% decrease in the search size."

Schaeffer (1994) reported a 5% decrease in the number of nodes searched when doubling the number of entries in the transposition table. It is remarkable that both authors arrive at effects of the same order of magnitude in spite of employing different move-ordering techniques.

We have tested the effect of doubling the number of positions in the transposition tables by conducting the experiments with eight different table sizes, viz. from 8K to 1024K positions[1].

6.4 Experimental

To test the ideas mentioned, an experiment has been conducted that observed the performance of every combination of the seven replacement schemes (with and without time stamping) and the eight table sizes. As stated in Section 5, the tests have been conducted on 18 positions, searching each position 3 to 7 ply; 8-ply searches have been performed for table sizes of 16K, 64K, 256K and 1024K, for a total of 11,088 observations. On one SUN 4 computer, this took an aggregate of roughly 1300 hours.

7. RESULTS

Some typical results will be graphically illustrated in this Section. When comparing the replacement schemes (see Figure 1) the table sizes have been kept constant; this implies that the three BIG schemes (BIG1, BIGALL, TWOBIG1) use slightly more memory and that the two-level schemes (TWOBIG1, TWODEEP) have half the number of entries of the other five schemes. It is claimed that these minor differences will not affect the interpretation of the results.

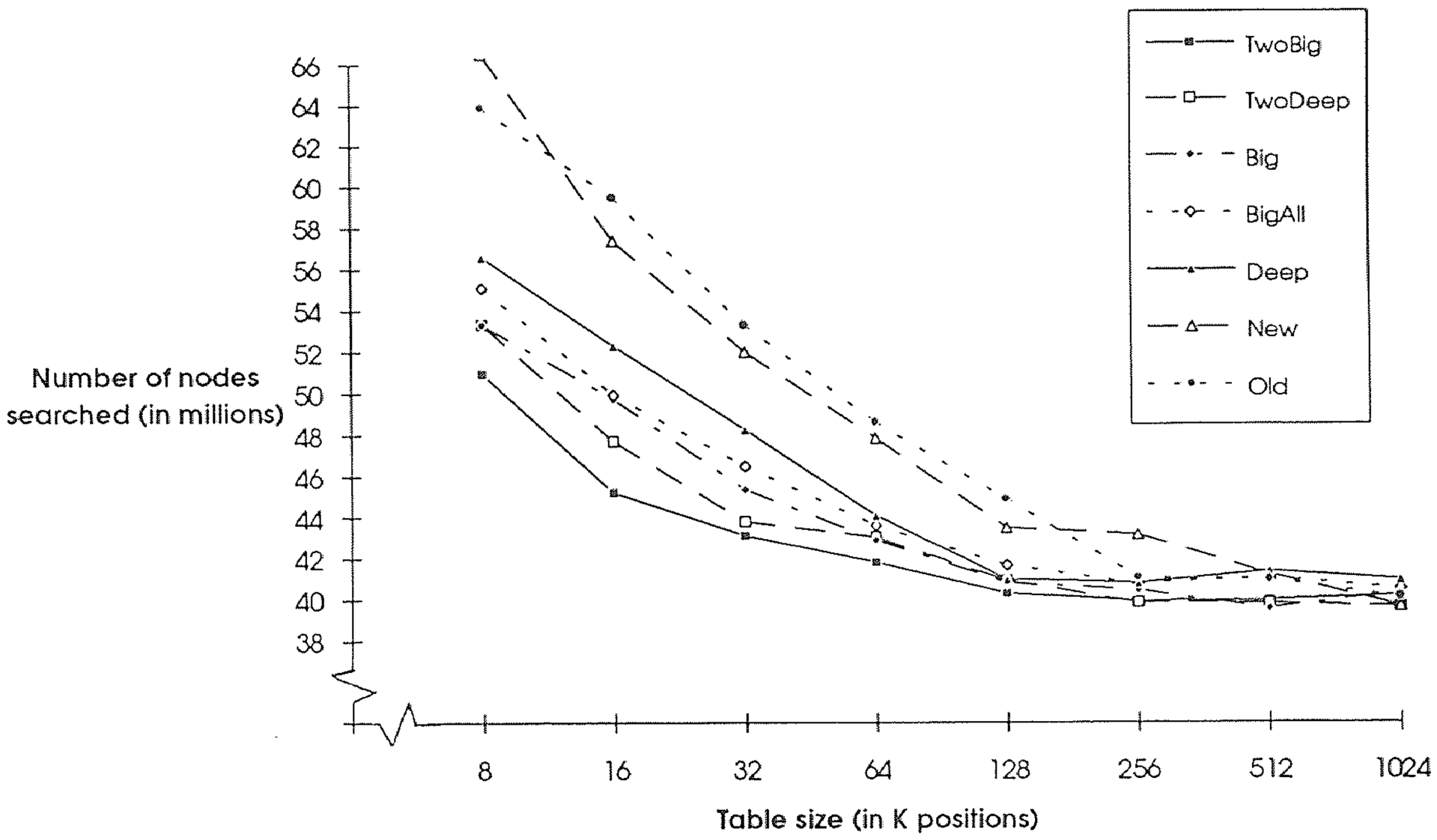

Figure 1: Effect of table size without time stamping, 7-ply searches.

[1] K positions is equal to 1024 positions.

The following trends seem to be evident in Figure 1. Any conclusions must be taken in the context of the small data set (18 positions) and the statistical variability.

- As table size increases, the number of nodes searched tends to constancy. In other words, at some point, possibly before 1024K in our case, no significant gains may be hoped for by increasing table size. This is caused by the larger percentage of tree nodes that can be retained in the transposition table: the probability of harmful collisions (i.e., collisions that cost many nodes) then greatly decreases.
- As table size increases, the spread between replacement schemes shrinks. For table sizes from 512K upwards, the spread is only around 3%, whereas the smallest practicable size, 8K, suggests a spread of no less than 23% between the best (TWOBIG1) and worst (NEW) scheme. This is a consequence of the argument above.
- The two-level-table schemes outperform those with one level only.
- Our data confirms Ebeling's (1986) statement, based on 10 positions, that TWODEEP "reduces search times by 5 to 10% for middle game positions" when compared with DEEP.

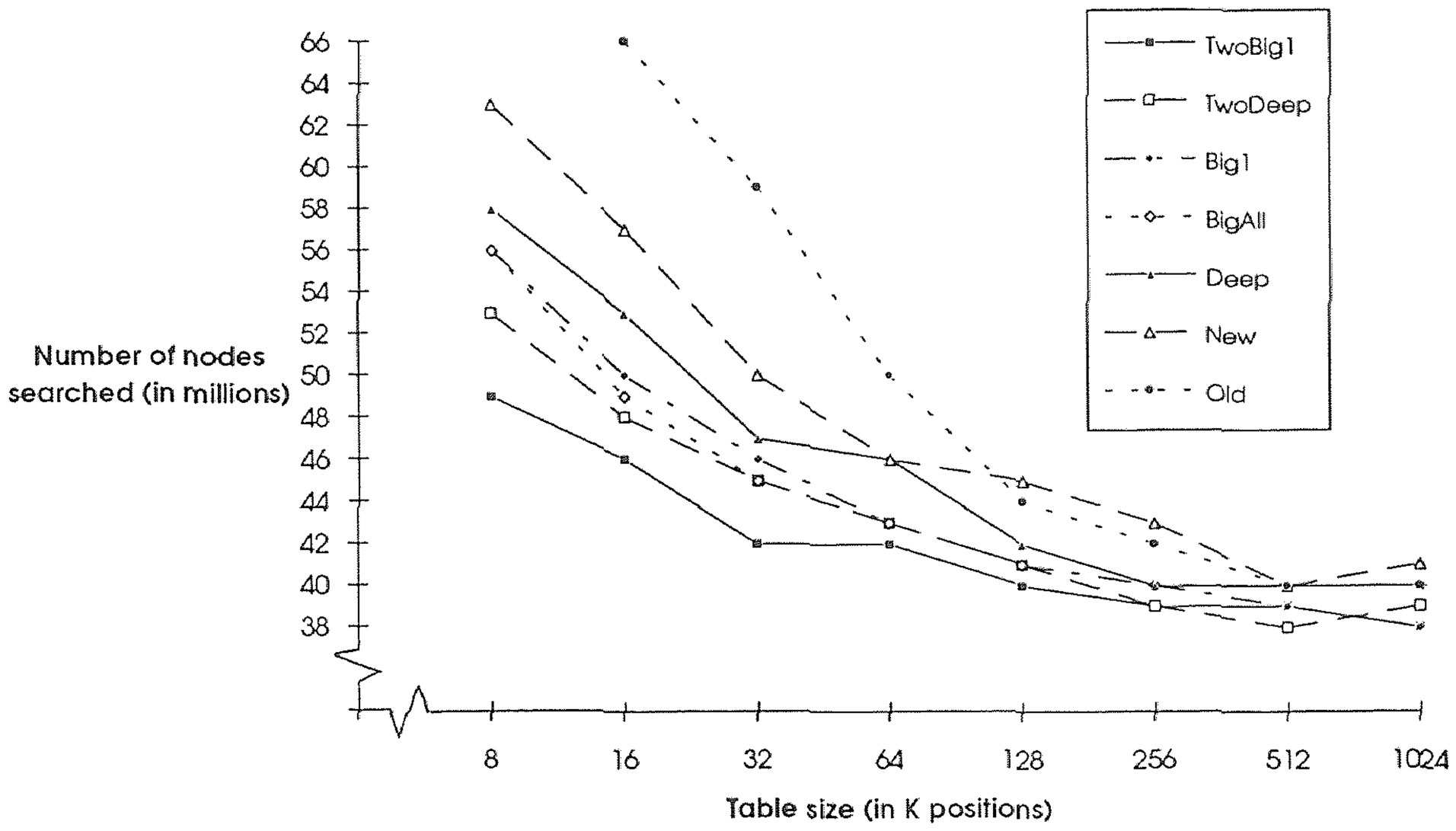

Figure 2: Effect of table size with time stamping, 7-ply searches.

Figure 2 has been included here, chiefly for comparison with Figure 1. Noting that the difference between them only is that Figure 2 shows results *with* time stamping as opposed to Figure 1 which represents clearing the table entries between searches, the following trends seem to be evident.

- *The shapes of all graphs are highly similar between the Figures 1 and 2.*
- The relative order of merit of the replacement schemes seems to be invariant for time stamping; whether one time stamps or clears the transposition tables between moves, TWOBIG1 appears to have a persistent edge.

At first sight time stamping seems to have a slight edge. Further research is needed to determine whether this improvement is significant. If so, it can be recommended since it only requires one additional bit per table position and requires little additional computation.

When limiting ourselves to a 3-ply search the use of a transposition table with time stamping is counterproductive in that it prolongs the search. The probable cause is an unfavourable move ordering, caused by a poor best-move suggestion from the transposition table. However, it is reassuring that the use of transposition tables is definitely advantageous at more realistic search depths of over 3 ply. An example is displayed in Figure 3.

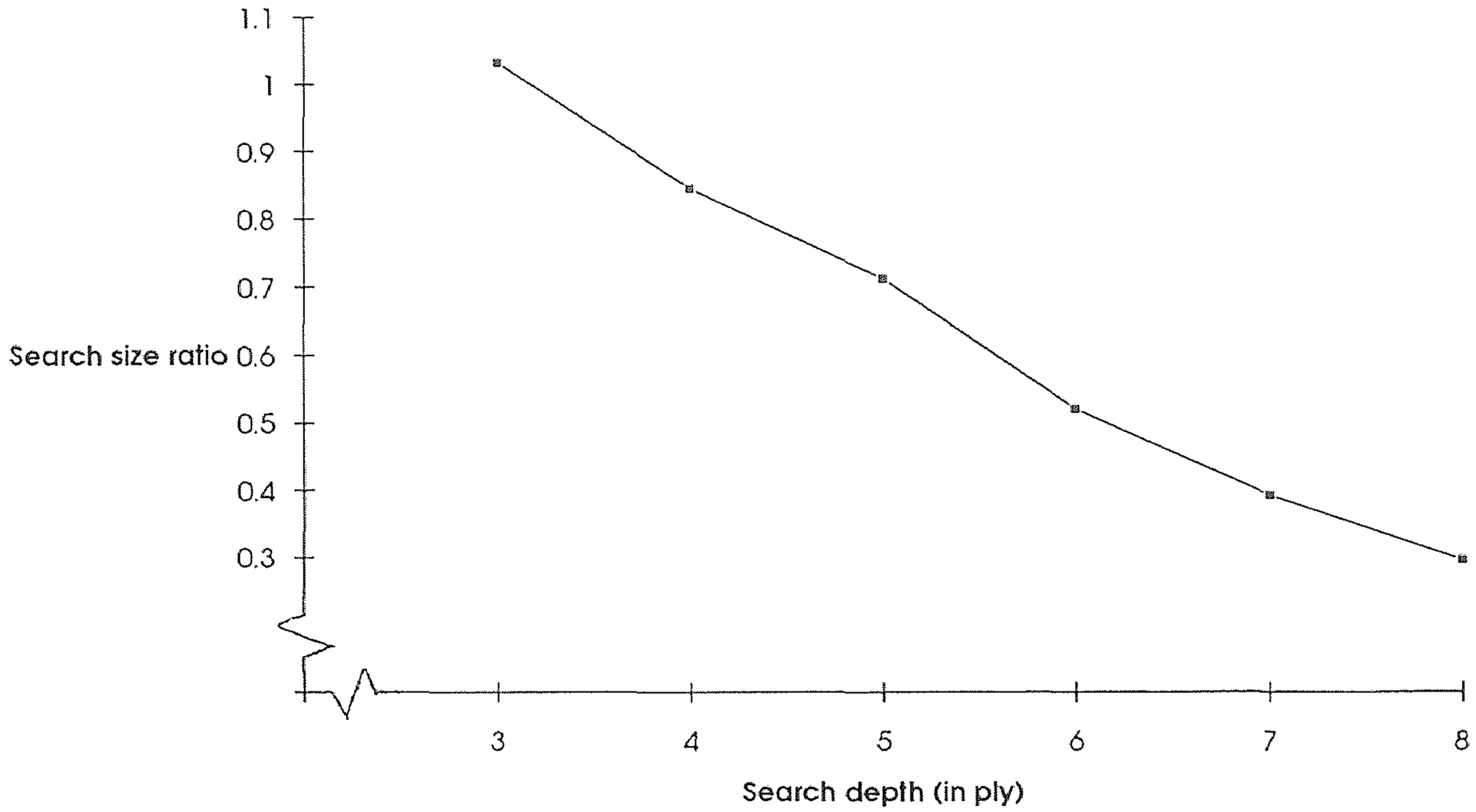

Figure 3: The effect of using a 1024K-positions transposition table (with time stamping in scheme TWOBIG1). The search size without transposition tables is 1.

8. CONCLUSIONS AND FUTURE RESEARCH

On logical grounds, one is tempted to conclude that the *number of nodes* of a subtree is a better estimate of the work performed (and therefore potentially to be saved) than the *depth* of that subtree. The experiments support this logic.
Taken at face value, the results favour a two-level scheme over any one-level scheme. If this is true, it would follow that DEEP, the most widely used scheme, is not best.

It must be stressed that the results have validity only as an exploration of methods and that the data on which they are based (18 consecutive positions from a single champion's game) may not be a statistically large enough sample. The first suggestion for further research therefore must be a repetition of these experiments on a much larger set of data. For this purpose, a uniform and portable platform is now available, in the form of ALIBABA.

9. ACKNOWLEDGEMENTS

This paper has gone through a number of provisional versions. We have profited from the many detailed criticisms, suggestions and hints by Professor Jonathan Schaeffer of the University of Alberta and we gladly acknowledge our debt to this expertise.

The present work is supported, in part, by the Foundation for Computer Science Research in the Netherlands (SION), with financial support from the Netherlands Organization for Scientific Research (NWO) (dossier code 612-22-306). The research has been performed in the framework of the SYRINX project (SYnthesis of Reliable Information using kNowledge of eXperts), part of a joint research effort of IBM and the University of Limburg (project code 561553).

10. REFERENCES

Akl, S.G. and Newborn, M.M. (1977). The Principal Continuation and the Killer Heuristic. *1977 ACM Annual Conference Proceedings*, pp. 466-473. ACM, Seattle.

Berliner, H.J. and Ebeling, C. (1989). Pattern Knowledge and Search: the SUPREM Architecture. *Artificial Intelligence*, Vol. 38, pp. 161-198. ISSN 0004-3702.

Ebeling, C. (1986). *All the Right Moves: A VLSI Architecture for Chess*. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, Pa. ISBN 0-262-05035-8.

Greenblatt, R.D., Eastlake, D.E., and Crocker, S.D. (1967). The Greenblatt Chess Program. *Proc. AFIPS Fall Joint Computer Conference 31*, pp. 801-810.

Hyatt, R.M., Gower, A.E., and Nelson, H.L. (1990). Cray Blitz. *Computers, Chess, and Cognition* (eds. T.A. Marsland and J. Schaeffer), pp. 111-132. Springer-Verlag, New York. ISBN 0-387-97415-6.

Knuth, D.E. and Moore, R.W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 6, pp. 293-326. ISSN 0004-3702.

Knuth, D.E. (1973). *The Art of Computer Programming. Vol. 3: Sorting and Searching*. Addison-Wesley Publishing Company, Reading, Massachusetts. ISBN 0-201-03803-X.

Kopec, D. and Bratko, I. (1982). The Bratko-Kopec Experiment: a Comparison of Human and Computer Performance in Chess. *Advances in Computer Chess 3* (ed. M.R.B. Clarke), pp. 57-72. Pergamon Press, Oxford. ISBN 0-08-026898-6.

Lang, K.J. and Smith, W.D. (1993). A Test Suite for Chess Programs. *ICCA Journal*, Vol. 16, No. 3, pp. 152-161. ISSN 0920-234X.

Marsland, T.A. (1986). A Review of Game-Tree Pruning. *ICCA Journal*, Vol. 9, No. 1, pp. 3-19. ISSN 0920-234X.

Nielsen, J.B. (1991). A Chess-Computer Test Set. *ICCA Journal*, Vol. 14, No. 1, pp. 33-37. ISSN 0920-234X.

Reinfeld, F. (1958). *Win at Chess*. Dover Publications, Inc., New York. ISBN 0-486-20438-3. Originally published (1945) as *Chess Quiz* by David McKay Company, New York.

Schaeffer, J. (1983). The History Heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16-19. ISSN 0920-234X.

Schaeffer, J. (1989). The History Heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 11, pp. 1203-1212.

Schaeffer, J. (1994). *Personal communication.*

Scott, J.J. (1969). A Chess-Playing Program. *Machine Intelligence 4* (ed. B. Meltzer and D. Michie), pp. 255-265. Edinburgh University Press.

Shannon, C.E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, Vol. 41, No. 7, pp. 256-275. Reprinted (1988) in *Computer Games I* (ed. D.N.L. Levy), pp. 81-88. Springer Verlag, New York. ISBN 0-387-96496-7.

Slate, J.S. and Atkin, L.R. (1983). CHESS 4.5: The Northwestern University Chess Program. *Chess Skill in Man and Machine* (ed. P.W. Frey), pp. 82-118. Springer-Verlag, New York. ISBN 0-387-90790-4.

Zobrist, A.L. (1970). *A New Hashing Method with Application for Game Playing*. Technical Report #88, Computer Science Department, The University of Wisconsin, Madison. Reprinted (1990) in *ICCA Journal*, Vol. 13, No. 2, pp. 69-73. ISSN 0920-234X.

APPENDICES

APPENDIX A The Test Positions

The test positions used for the experiments are the WTM positions from move 15 onwards.

Kasparov - Short
Euwe Memorial VSB tournament, Round 2, Amsterdam, 1994
1. e4 e6 2. d4 d5 3. Nc3 Nf6 4. e5 Nfd7 5. f4 c5 6. Nf3 Nc6 7. Be3 cxd4 8. Nxd4 Bc5 9. Qd2 0-0 10. 0-0-0 a6 11. h4 Nxd4 12. Bxd4 b5 13. Rh3 b4 14. Na4 Bxd4 15. Qxd4 f6 16. Qxb4 fxe5 17. Qd6 Qf6 18. f5 Qh6 19. Kb1 Rxf5 20. Rf3 Rxf3 21. gxf3 Qf6 22. Bh3 Kf7 23. c4 dxc4 24. Nc3 Qe7 25. Qc6 Rb8 26. Ne4 Nb6 27. Ng5 Kg8 28. Qe4 g6 29. Qxe5 Rb7 30. Rd6 c3 31. Bxe6 Bxe6 32. Rxe6 1-0

APPENDIX B Some Results

A full set of our results is available upon request from the first author. Below, we present a selection of extreme cases, viz. those without the use of a transposition table and those with a transposition table of the maximal size investigated, 1024K. The latter is presented without time stamping as well as with it. In the former case the transposition tables are cleared between moves. All figures reported are number of nodes visited in thousands.

3 ply	4 ply	5 ply	6 ply	7 ply	8 ply
185	1,083	3,847	23,573	96,969	610,696

Table 1: Results without a transposition table.

scheme	3 ply	4 ply	5 ply	6 ply	7 ply	8 ply
TwoBig1	194	897	2,791	12,591	40,161	191,687
TwoDeep	194	897	2,791	12,507	39,596	185,354
Big1	194	907	2,788	12,568	40,324	197,502
BigAll	194	907	2,788	12,542	40,478	197,377
Deep	194	899	2,816	13,206	40,916	196,826
New	194	899	2,816	12,702	39,634	214,693
Old	194	931	2,798	12,897	40,116	202,986

Table 2: Results with a 1024K-positions transposition table without time stamping.

scheme	3 ply	4 ply	5 ply	6 ply	7 ply	8 ply
TwoBig1	191	915	2,738	12,296	38,062	180,187
TwoDeep	191	915	2,738	11,989	38,514	191,747
Big1	191	914	2,764	12,327	38,357	197,978
BigAll	191	914	2,792	11,822	39,985	197,573
Deep	191	907	2,753	11,887	39,622	195,008
New	191	907	2,753	12,112	40,656	202,209
Old	191	934	2,777	11,926	39,580	205,200

Table 3: Results with a 1024K-positions transposition table with time stamping.