Information in Transposition Tables

Article · February 1970

Source: CiteSeer

CITATIONS
13

READS
514

3 authors:

Dennis Breuker
Hogeschool Arnhem and Nijmegen
22 PUBLICATIONS **315** CITATIONS
SEE PROFILE

Jos W. H. M. Uiterwijk
Maastricht University
175 PUBLICATIONS **2,372** CITATIONS
SEE PROFILE

H. Jaap Van Den Herik
Leiden University
607 PUBLICATIONS **5,523** CITATIONS
SEE PROFILE

Information in Transposition Tables

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

University of Limburg
The Netherlands

ABSTRACT

Due to transpositions a search tree can be considered as a search graph. The transpositions are stored with information about previous searches in a transposition table. Although the use of transposition tables is standard practice, it is still an open question how large the overall reduction is and especially which information has the largest impact on the reduction.
The paper describes an experiment to distinguish between the reduction given the best move is stored or the value of the best move is stored. A second experiment compares storing the bound values for minimal-window search with storing the true values. It transpires that the highest reduction comes from re-using bounds because they adequately generate cutoffs.
Since we know that from a certain transposition-table size not much is to be gained from doubling the number of entries, it might be useful to store additional information to see whether this leads to a better result. Our first experiments on the use of more information per entry versus more entries indicate that additional memory can better be used for storing additional information than for doubling the size of the transposition table.

1 Two Problem Statements

When searching for a move chess programs build large search *trees*. Since a position sometimes can be arrived at by several distinct move sequences the size of the search tree can be reduced considerably if previous results for a position

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

already encountered are still available. The information can be stored in a large direct-access table, called the *transposition table* (Greenblatt *et al.*, 1967; Slate and Atkin, 1977). A closer inspection shows that the search tree now can be considered as a search *graph*, due to the transpositions.

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

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 only a few pieces on the board. Still, it is an open question how large the overall reduction is and especially which information has the largest impact on the reduction. In this paper we quantify the merit of the different transposition-table components. Moreover, we address the general question how we can profit most efficiently from additional memory: by enlarging the transposition table or by adding additional information?

In Section 2 the traditional components of a transposition-table entry are listed and a series of experiments for quantifying their respective merits is given. The experiments are described in Section 3. Section 4 contains additional components for a transposition table. In Section 5 the results of a second set of experiments are given. Preliminary conclusions are provided in Section 6.

2 Traditional Components

For an entry in a transposition table to be effective, it should at least contain the following information (Marsland, 1986; Hyatt *et al.*, 1990):

key [2]: contains the more significant bits of the hash value, viz. the complement of the bits used for the hash index. The key distinguishes among different board positions having the same hash index.

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

move : contains the best move in the position, which either caused a cutoff, or obtained the highest score. The move is used for move ordering.

score : contains the value 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. The score is used to adjust the α and β bounds of the search.

flag : contains information on the score. The flag indicates whether the score is a true value, an upper bound or a lower bound.

searchDepth : contains 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 search depth is $n - m$. The searchDepth indicates how deep a previously encountered position has been investigated.

We call a transposition table with these five information fields a traditional table. During the search, each position is looked up in the table. If the position is found the information serves one of three purposes, depending on the contents of **flag** and **searchDepth**.

1. The depth still to be searched is less than or equal to the depth retrieved from the table *and* the retrieved score is a true value. No further search has to be done: the search value is retrieved from the table.[3] Usually, the best move is also retrieved from the table, and used for determining the principal variation.

2. The depth still to be searched is less than or equal to the depth retrieved from the table *and* the retrieved score is *not* a true value. In this case the score can be used to adjust either the α value (if the score is a lower bound) or the β value (if the score is an upper bound). If this causes α to be greater than or equal to β, then a cutoff occurs and no search has to be done. Otherwise, the retrieved move can be used as a first candidate, since it was considered best (or at least good enough to yield a cutoff) previously.

3. The depth still to be searched is greater than the depth retrieved from the table. In this case only the retrieved move is used[4]. It can be investigated first, since it was considered best for a shallow search, the probability being high that it also will be best for deeper searches. Thus the move is used to improve move ordering.

[3] If the depth still to be searched is less than the depth retrieved, the search results may differ from the results when searching without a transposition table.

[4] Programs using aspiration search also use the score. It is used for setting the search window (Brudno, 1963; Berliner, 1974; Gillogly, 1978).

In summary, a transposition table is used for two purposes: (1) the retrieved move is used for move ordering, and (2) the retrieved score is used for establishing the value of the position. In the last case the score is either a true value, and the position does not have to be researched, or a bound value, in which case either the α value or the β value is adjusted.

For investigating the merits of move vs. value of move we have performed six experimental searches.

1. Search without a transposition table.
2. Search with a traditional transposition table, without **score**.
3. Search with a traditional transposition table, without **move**.
4. Search with a traditional transposition table, without **move**, only using the score information if the score is a **true value**.
5. Search with a traditional transposition table, without **move**, only using the score information if the score is a **bound value**.
6. Search with a traditional transposition table, with **score**, using the score information both if the score is a **true value** or a **bound value**, and **move**.

The experiments 1 and 6 are performed for completeness' sake.

3 Experiments with a traditional table

The test set used for the experiments consists of 18 consecutive WTM (White-to-move) positions taken from a middle game and 21 consecutive WTM positions taken from an endgame (see Appendix). Both games are played by human experts. The 18 middle-game positions have been searched for 7 ply, the 21 endgame positions for 10 ply. The replacement scheme used for all experiments is TWOBIG1, the scheme which performs best (Breuker *et al.*, 1994). The scheme is implemented as a two-level table, potentially storing two positions per entry. Upon a collision the newest position is always stored, overwriting the less important one of the previous two positions (in terms of number of nodes searched). All experiments have been performed with a series of transposition tables, ranging from 8K entries to 256K entries. The search algorithm used is minimal-window, iterative-deepening α-β search. As the measure for quantifying search effort we use the number of *all* nodes investigated, i.e., summing up interior nodes and leaf nodes. The number of nodes in the Figures 1 and 2

Figure 1: The use of a transposition table in the middle game; 7-ply searches.

Figure 1 clearly shows that the use of a transposition table (experiment 6) is profitable in terms of number of nodes searched compared to searching without a transposition table (experiment 1). Further, the use of the **score** of a transposition (experiment 3) is more profitable than the use of the **move** (experiment 2). This is due to the fact that whenever one of the bounds of the minimal window is updated, its lower bound will be greater than its upper bound, thereby causing a cutoff. It appears that whenever a position is found in the transposition table, the retrieved score causes a cutoff in about 50% of the cases. However, this effect stems fully from bound values (experiment 5). True values (experiment 4) hardly have any effect in this respect, or, even worse, have a

Figure 2: The use of a transposition table in the endgame; 10-ply searches.

The results of the experiments on the endgame positions are analogous to the results of the experiments on the middle-game positions, but are more pronounced as can be seen in Figure 2. Moreover, the use of a transposition table is more profitable in endgames than in middle games. Further, with the largest transposition table used (256K entries) in middle games using only the **move** from the table results in about a 33% node decrease, whereas in the endgame the decrease is about 66%. If in addition the **score** from the table is used, a total decrease of about 60% in the middle game and about 90% in the endgame is obtained.

The experiments reported in this section re-establish the benefit of using transposition tables, especially in endgame positions. The largest profit stems from storing bound values.

4 Additional Components

In our search for storing additional information in a transposition-table entry we have found several suggestions, among others made by Schaeffer (1994) and Stanback (1994). From their suggestions, we mention five additional components:

date : contains the root's ply number in the game when the position was stored. Sometimes only a 1-bit date flag is used, stating if the position is from an 'old' search or not. The date is used for time-stamping (Breuker *et al.*, 1994). A position will be overwritten by a position with a newer date.

depth : contains the number of ply seen from the root. A position is more important if it is nearer to the root, since it has a higher probability of being re-searched; possible savings are most likely larger than savings for positions deeper in the tree.

extension : contains a boolean value, denoting if a search extension was done at this position. The extension criteria at a node may vary (e.g., because the extension is dependent on the α-β window), resulting in an extension one time and *not* in an extension the other time. The boolean extension helps to overcome this problem (which is especially important when doing a re-search).

principal : contains a boolean value, denoting if this position is part of the principal variation of a child of the root[5]. Positions which are part of the principal variation of a root's child are important positions, and may not be overwritten by other positions.

draw : contains a boolean value, denoting if the backed-up score of the position is a proven draw. This is useful when you want to distinguish between variations resulting in positions which are real draws, and variations resulting in balanced positions (which obtain a draw score).

Presumably this additional information will have an impact on the number of nodes searched. However, only very few researchers have published even preliminary results about experiments on these additional components. Breuker *et al.* (1994) mention an experiment testing the use of a 1-bit date flag, concluding that "time stamping seems to have a slight edge". In general it seems that adding these new components to an entry is not very profitable (Schaeffer, 1996).

Storing the additional information described above does not take up much memory. Most fields need one bit of storage only, since they are booleans. The choice

[5]Note that this is a way to implement the refutation table using the transposition table.

for small additional components is made on purpose, since a larger entry results in a transposition table with fewer entries (assuming the same amount of memory is available). However, Breuker *et al.* (1994) showed that from a certain transposition-table size not much is to be gained from doubling the number of entries. Moreover, if the available memory is less than the memory needed for doubling the number of entries, it still can be used for storing more information in an entry.

The above considerations lead to the question how to use additional fields, taking up more memory than only one bit. Instead of storing the best move (which can be seen as a 1-ply principal variation) in a transposition-table entry, it may be interesting to investigate the effects of storing a deeper *principal variation* in an entry. This principal variation (PV) can be used to guide the search. If a position is not present in the transposition table, a good move may still be available from the n-ply PV information of an ancestor position. In the next section preliminary experiments are given investigating the effects of storing an n-ply PV in a transposition-table entry.

5 Experiments with Storing the PV

The conditions for the experiments are equal to the conditions mentioned in Section 3. As a measure for quantifying search effort we use the number of *all* nodes investigated, i.e., summing up interior nodes and leaf nodes. Again, the number of nodes in the Figures 3 and 4 are the cumulative results of all 18 and 21 positions, respectively. We have tested the results of storing an n-ply PV (n = 2..5) in an entry versus storing only the best move (a 1-ply PV).

5.1 Middle-Game Experiments

In Figure 3 the results of the PV experiments on middle-game positions are depicted. The number of nodes investigated are shown as a function of the transposition-table size.

Our first observation is that storing n-ply variations seems hardly worthwhile: the effects are small and severely dependent on the size of the transposition table[6]. The explanation for this is that for less than 0.1% of the nodes investigated a position appears to be absent in the transposition table, whereas a PV from an ancestor still is available. To give at least some quantification, it can be seen that with the largest transposition table (256K entries), storing a

[6]Note the small vertical scale.

Figure 3: Storing an n-ply variation in the middle game; 7-ply searches.

5-ply variation instead of a 1-ply variation wins roughly 3%, only marginally outperforming the 1% gain by simply doubling the table size to 512K entries (Breuker *et al.*, 1994).

5.2 Endgame Experiments

The results of the experiments on the endgame positions are analogous to the results of the experiments on the middle-game positions, as can be seen in Figure 4. Here again, for the largest transposition-table size, the 5-ply variation outperforms the 1-ply variation, but this time by some 12%.

6 Conclusions

We have described two series of experiments on the use of a transposition table. First, it is examined which information is more important to store in a transposition-table entry: the best move in a position, or the score of that move. It follows that storing the score of a position is more profitable than storing the best move. This result holds for middle-game positions as well as for

Figure 4: Storing an n-ply variation in the endgame; 10-ply searches.

endgame positions. Second, it was found that for minimal-window search bound values had a much larger effect than true values. This effect, although nowadays expected, contrasts the idea from which the transposition tables originally were devised, i.e., avoiding the re-search of positions searched fully before.

Furthermore, we have tested the effect of storing an n-ply PV ($n = 2..5$) in an entry, instead of only the best move (a 1-ply PV). Preliminary results show that a 5-ply variation may win roughly 3% for the middle game, to 12% for the endgame, though more experiments will be needed to validate this result.

Based on the above experiments, for future research it is recommended to concentrate on storing additional information which affects the number of cutoffs generated by bound values.

References

[1] Berliner H.J. and Ebeling C. (1989). Pattern Knowledge and Search: the SUPREM Architecture. *Artificial Intelligence*, Vol. 38, pp. 161–198.

[2] Berliner H.J. (1974). *Chess as Problem Solving: The Development of a Tactics Analyzer.* Ph.D. thesis, Carnegie-Mellon University, Pittsburgh,

Pennsylvania.

[3] Breuker D.M., Uiterwijk J.W.H.M., and Herik H.J. van den (1994). Replacement Schemes for Transposition Tables. *ICCA Journal*, Vol. 17, No. 4, pp. 183–193.

[4] Brudno A.L. (1963). Bounds and Valuations for Abridging the Search of Estimates. *Problems of Cybernetics*, Vol. 10, pp. 225–241. Translation of Russian original in *Problemy Kibernetiki*, Vol. 10, May 1963, pp. 141-150.

[5] Ebeling C. (1986). *All the Right Moves: A VLSI Architecture for Chess.* Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[6] Gillogly J.J. (1978). *Performance Analysis of the Technology Chess Program.* Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[7] Greenblatt R.D., Eastlake D.E., and Crocker S.D. (1967). The Greenblatt Chess Program. *Proceedings of the AFIPS Fall Joint Computer Conference 31*, pp. 801–810. Reprinted (1988) in *Computer Chess Compendium* (ed. D.N.L. Levy), pp. 56-66. B.T. Batsford Ltd, London.

[8] 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. 111–130. Springer-Verlag, New York.

[9] Knuth D.E. (1973). *The Art of Computer Programming. Volume 3: Sorting and Searching.* Addison-Wesley Publishing Company, Reading, Massachusetts.

[10] Marsland T.A. (1986). A Review of Game-Tree Pruning. *ICCA Journal*, Vol. 9, No. 1, pp. 3–19.

[11] 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.

[12] Schaeffer J. (1994). *Personal communication.*

[13] Schaeffer J. (1996). *Personal communication.*

[14] Slate J.D. and Atkin L.R. (1977). 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. Second Edition, 1983.

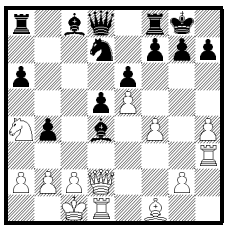
[15] Stanback J.S. (1994). *Personal communication.*

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

Appendix: The Test Positions

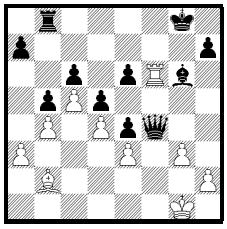
The middle-game experiments use the 18 WTM positions from move 15 onwards of:

Kasparov–Short, Amsterdam (round 2) 1994
1. e4 e6 2. d4 d5 3. ♘c3 ♘f6 4. e5 ♘fd7 5. f4 c5 6. ♘f3 ♘c6 7. ♗e3 cxd4 8. ♘xd4 ♗c5 9. ♕d2 0–0 10. 0–0–0 a6 11. h4 ♘xd4 12. ♗xd4 b5 13. ♖h3 b4 14. ♘a4 ♗xd4 15. ♕xd4 f6 16. ♕xb4 fxe5 17. ♕d6 ♕f6 18. f5 ♕h6+ 19. ♔b1 ♖xf5 20. ♖f3 ♖xf3 21. gxf3 ♕f6 22. ♗h3 ♔f7 23. c4 dxc4 24. ♘c3 ♕e7 25. ♕c6 ♖b8 26. ♘e4 ♘b6 27. ♘g5+ ♔g8 28. ♕e4 g6 29. ♕xe5 ♖b7 30. ♖d6 c3 31. ♗xe6+ ♗xe6 32. ♖xe6 1–0


Position after 14..Bxd4

The endgame experiments use the 21 WTM positions from move 34 onwards of:

Rabinovich–Romanovsky, Leningrad championship 1934
1. c4 ♘f6 2. ♘c3 c6 3. d4 d5 4. ♘f3 ♘e4 5. e3 e6 6. ♗d3 f5 7. ♕c2 ♘d7 8. b3 ♗b4 9. ♗b2 ♕a5 10. ♖c1 0–0 11. 0–0 ♗d6 12. ♘e2 ♕d8 13. ♘e5 ♕h4 14. f3 ♘ec5 15. g3 ♕h6 16. ♘f4 ♘xd3 17. ♘exd3 g5 18. ♘g2 ♘f6 19. ♖ce1 g4 20. fxg4 ♘xg4 21. ♘gf4 ♘f6 22. ♖e2 ♖f7 23. b4 ♘e4 24. ♘c5 ♖b8 25. a3 b6 26. ♘xe4 fxe4 27. ♖ef2 ♗d7 28. c5 ♗xf4 29. ♖xf4 ♖xf4 30. ♖xf4 b5 31. ♕f2 ♗e8 32. ♖f6 ♗g6 33. ♕f4 ♕xf4 34. ♖xf4 h5 35. h3 ♔g7 36. ♗c3 ♗f5 37. g4 hxg4 38. hxg4 ♗g6 39. ♔g2 ♔h6 40. ♖f6 ♖e8 41. ♗e1 ♔g7 42. ♖f1 ♖a8 43. ♔h3 a6 44. ♗g3 ♖h8 45. ♗h4 ♖f8 46. ♖xf8 ♔xf8 47. ♗g3 e5 48. ♗xe5 ♔f7 49. ♔h4 ♔e6 50. ♔g5 ♗e8 51. ♔h6 ♗f7 52. ♔g7 ♗e8 53. g5 ♔f5 54. ♔f8 1–0



Position after 33..Qxf4