The Relative History Heuristic

Mark H.M. Winands, Erik C.D. van der Werf,
H. Jaap van den Herik, and Jos W.H.M. Uiterwijk

Department of Computer Science, Institute for Knowledge and Agent Technology,
Universiteit Maastricht, P.O. Box 616
6200 MD Maastricht, The Netherlands
{m.winands, e.vanderwerf, herik, uiterwijk}@cs.unimaas.nl

Abstract. In this paper a new method is described for move ordering, called the relative history heuristic. It is a combination of the history heuristic and the butterfly heuristic. Instead of only recording moves which are the best move in a node, we also record the moves which are applied in the search tree. Both scores are taken into account in the relative history heuristic. In this way we favour moves which on average are good over moves which are sometimes best. Experiments in LOA show that our method gives a reduction between 10 and 15 per cent of the number of nodes searched. Preliminary experiments in Go confirm this result. The relative history heuristic seems to be a valuable element in move ordering.

1 Introduction

Most modern game-playing computer programs successfully use $\alpha\beta$ search [10]. The efficiency of $\alpha\beta$ search is dependent on the enhancements used [4]. Move ordering is one of the main techniques to reduce the size of the search tree. There exist several move-ordering techniques, which can be qualified by their dependency on the search algorithm [11]. *Static* move ordering is independent on the search. These techniques rely on game-dependent knowledge. The ordering can be acquired by using expert knowledge (e.g., favouring capture moves in Chess) or by learning techniques (e.g., the Neural MoveMap Heuristic [12]). *Dynamic* move ordering is dependent on the search. These techniques rely on information gained during the search. The transposition-table move [3], the killer moves [1], and the history heuristic [15] are well-known examples.

The history heuristic is a popular choice for ordering moves dynamically, in particular when other techniques are not applicable. In the past the butterfly heuristic [8] was proposed to replace the history heuristic, but it did not succeed. In this paper we propose a new dynamic move-ordering variant, called the relative history heuristic, to replace the history heuristic. The idea is that besides the number of times a move was chosen as a best move, we also record the number of times a particular move was explored.

This paper is organised as follows. In Section 2 we review the history heuristic and the butterfly heuristic. Next, the relative history heuristic is introduced in

Section 3. The test environment is described in Section 4. Subsequently, the results of the experiments are given in Section 5. Finally, in Section 6 we present our conclusion and propose topics for future research.

2 The History Heuristic and the Butterfly Heuristic

The history heuristic is a simple, inexpensive way to reorder moves dynamically at interior nodes of search trees. It was invented by Schaeffer [15] and has been adopted in several game-playing programs. Unlike the killer heuristic, which only maintains a history of the one or two best killer moves at each ply, the history heuristic maintains a history for every legal move seen in the game tree. Since there is only a limited number of legal moves, it is possible to maintain a score for each move in two (black and white) tables. At every interior node in the search tree the history-table entry for the best move found is incremented by a value (e.g., 2^d, where d is the depth of the subtree searched under the node). The best move is in this case defined as the move which either causes an alpha-beta cut-off, or which causes the best score. When a new interior node is examined, moves are re-ordered by descending order of their scores. The scores in the tables can be maintained during the whole game. Each time a new search is started the scores are decremented by a factor (e.g., divided by 2). They are only reset to zero or to some default values at the beginning of a complete new game. Details on the effectiveness or the strategy to maintain history scores during the whole game are dependent on the domain or game program.

The history heuristic does not cost much memory. The history tables are defined as two tables with 4096 entries (64 from_squares $\times$ 64 to_squares), where each entry is 4 or 8 bytes large. These tables can be easily indexed by a 12-bit key representing the origin and destination. In the history table we have also defined moves which are illegal.

Hartmann [8] called attention to two disadvantages of the history heuristic. (A) Quite some space for the history table is wasted, because space for illegal moves is reserved too. For instance, in the game of Chess 44 per cent of the possible moves are legal. In LOA, this number is even lower because knight moves are not allowed. This gives that 1456 of the 4096 moves are legal, meaning that only 36 per cent of the entries in the table are used. Although this waste of memory is not a problem for games with a small dimensionality of moves, it can be a problem for games with a larger dimensionality of moves (for instance Amazons). (B) Moreover, Hartmann pointed out that some moves are played less frequently than others. There are two reasons for this. (B1) The moves are less frequently considered as good moves. (B2) The moves occur less frequently as legal moves in a game. The disadvantage of the history heuristic is that it is biased towards moves that occur more often in a game than others. However, the history heuristic has as implicit assumption that all the legal moves occur roughly with the same frequency in the game (tree). So, in games where this condition approximately holds an absolute measure seems appropriate. But in other games where some moves occur more frequently than other moves, we

should resort to other criteria. For instance, assume we have a move which is quite successful when applicable (e.g., it then causes a cut-off) but it does not occur so often as a legal move in the game tree. This move will not obtain a high history score and is therefore ranked quite low in the move ordering. Therefore a different valuation of such a move may be considered.

To counter some elements of the two disadvantages Hartmann [8] proposed an alternative for the history heuristic, the butterfly heuristic. This heuristic takes the move frequencies in search trees into account. Two tables are needed (one for Black and one for White), called *butterfly boards*. They are defined in the same way as in the history heuristic (i.e., 64 from_squares × 64 to_squares). Any move that is not cut is recorded. Each time a move is executed in the search tree, its corresponding entry in the butterfly board (for each side) is also incremented by a value. Moves are now reordered by their butterfly scores. The butterfly heuristic was denied implementation by its inventor, since he expected that it would be far less effective than the history heuristic.

3 The Relative History Heuristic

We believe that we can considerably improve the performance of the history heuristic in some games by making it *relative* instead of absolute. The score used to order the moves (*movescore*) is given by the following formula:

$$movescore = \frac{hhscore}{bfscore} \tag{1}$$

where *hhscore* is the score found in the history table and *bfscore* is the score found in the butterfly board. We call this move ordering the relative history heuristic. We remark that we only *update* the entries of moves seen in the regular search, not in the quiescence search, because some (maybe better) moves are not investigated in the quiescence search. We *apply* the relative history heuristic everywhere in the tree (including in the quiescence search) for move ordering.

In some sense this heuristic is related to the realization-probability search method [19]. In that scheme the move frequencies gathered *offline* are used to limit or extend the search.

4 Test Environment

In this section the test environment is described which is used for the experiments. First, we briefly explain the game of Lines of Action (LOA). Next, we give some details about the search engine MIA.

4.1 Lines of Action

Lines of Action (LOA) [14] is a two-person zero-sum chess-like connection game with perfect information. It is played on an 8×8 board by two sides, Black and

White. Each side has twelve pieces at its disposal. The black pieces are placed in two rows along the top and bottom of the board, while the white pieces are placed in two files at the left and right edge of the board. The players alternately move a piece, starting with Black. A move takes place in a straight line, exactly as many squares as there are pieces of either colour anywhere along the line of movement. A player may jump over its own pieces. A player may not jump over the opponent's pieces, but can capture them by landing on them. The goal of a player is to be the first to create a configuration on the board in which all own pieces are connected in one unit. In the case of simultaneous connection, the game is drawn. The connections within the unit may be either orthogonal or diagonal. If a player cannot move, this player has to pass. If a position with the same player to move occurs for the third time, the game is drawn.

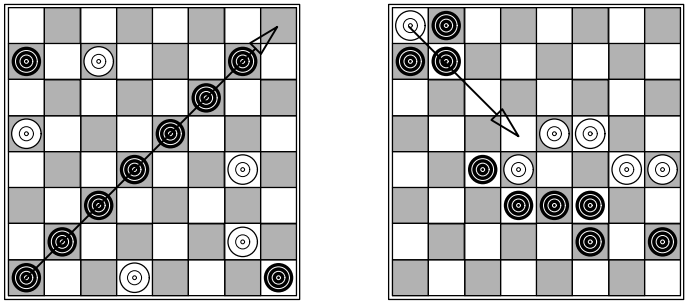

Fig. 1. (a) Rare move; (b) Blocked move.

LOA is a nice test bed for the relative history heuristic. Dependent on the position some moves occur more often than others in the search tree. For example, moves going seven squares far are possible if and only if there are seven pieces of the same colour side by side on that line. In Figure 1a it is possible to move **a1-h8**, but it is very rare that in a real game a position occurs where seven pieces of the same colour occupy a diagonal. In contrast, consider a move like **a8-d5** which occurs regularly in a game, but in the position depicted in Figure 1b it will not often be applied in the corresponding search tree, since most of the time Black will not consider to move its piece on **b7**.

4.2 MIA IV

MIA IV is a LOA-playing tournament program, which won the 8^{th} Computer Olympiad. It performs an $\alpha\beta$ depth-first iterative-deepening search. Several techniques are implemented to make the search efficient. The program uses PVS (Principal Variation Search) to narrow the $\alpha\beta$ window as much as possible [13]. A transposition table with 2^{21} double entries (using the *two-deep* replacement scheme [3]) is applied to prune a subtree or to narrow the $\alpha\beta$ window. At all

interior nodes which are more than 2 ply away from the leaves, the program generates all the moves to perform the Enhanced Transposition Cutoffs (ETC) scheme [18]. Next, a null move [7] is performed before any other move and it is searched to a lower depth (reduced by R) than other moves. In the search tree we distinguish three types of nodes, namely PV nodes, CUT nodes, and ALL nodes [10, 13]. The null move is done at CUT nodes *and* at ALL nodes. At a CUT node a variable scheme, called adaptive null move [9], is used to set R. If the remaining depth is more than 6, R is set to 3. When the number of pieces of the side to move is lower than 5 the remaining depth has to be more than 8 to set R to 3. In all other cases R is set to 2. For ALL nodes $R = 3$ is used. If the null move does not cause a β cut-off, multi-cut [2] is performed. Experiments showed that using multi-cut is not only beneficial at CUT nodes but also at ALL nodes [22]. For move ordering, the move stored in the transposition table, if applicable, is always tried first. Next, two killer moves [1] are tried. These are the last two moves, which were best or at least caused a cut-off at the given depth. Thereafter follow: (1) capture moves going to the inner area (the central 4×4 board) and (2) capture moves going to the middle area (the 6×6 rim). All the other moves are ordered decreasingly according to their scores in the (relative) history table [15]. If a cut-off occurs because of multi-cut, transposition table or ETC, its corresponding entry in the history table or butterfly board is also updated with the depth used for exploration. In the leaf nodes of the tree a quiescence search is performed. This quiescence search looks at capture moves that form or destroy connections [21] and at capture moves going to the central 4×4 board.

5 Experiments

In this section we show the results of various experiments with the relative history heuristic. This is done on a test set of 171 LOA positions.[1] We performed six series of experiments. In the first and second series, we tested the standard history heuristic and the relative history heuristic with different increments, respectively (Subsection 5.1). In the third, fourth, and fifth series, we compared the performance of the relative history heuristic with the standard history heuristic under different configurations (Subsection 5.2). Finally, in the sixth series of experiments, we tested the performance of the relative history heuristic in another domain, namely for 24 test positions for 6×6 Go (Subsection 5.3).

5.1 Increment Settings

In the following two series of experiments we tried to find the optimal increment setting for the history table and the butterfly board, which gave the largest node reduction. Using our set of 171 LOA positions, the program was tested for depth 14 using its normal enhancements as described in the previous subsection.

In the first series of experiments we tested the increments of the history table in a configuration where we used the standard history heuristic. Initially

[1] The test set can be found at http://www.cs.unimaas.nl/m.winands/loa/TMP.zip.

Table 1. Performance of the history heuristic with different increments on a test set of 171 positions.

History Increment	Total Nodes		
	depth 5	depth 9	depth 14
0	2,480,001	188,717,928	30,997,625,767
1	1,901,956	113,163,113	14,478,291,866
d	1,896,429	111,283,177	14,064,388,392
d^2	1,900,055	111,673,124	13,915,673,199
2^d	1,878,114	111,471,652	13,925,222,389

the history heuristic was developed for programs that were searching to a depth much less than 14. Considering the nature of a search tree, it might be that the best increment to be used depends on the search depth. Therefore we performed experiments for the original depths 5 and 9 used as test depths by Schaeffer [15, 17]. The following increments were used: 1, d, d^2, and 2^d, where d is the explored depth of the move causing the cut-off. The increment 2^d is the standard increment of the history table. The result of the search without history heuristic (increment 0) is given for comparison. Table 1 shows that there is not much difference between the size of the search tree using increments d, d^2 and 2^d. For depth 14, the history heuristic gives in all the cases a reduction of approximately 55 per cent of the number of nodes searched. Unlike data for Chess, reported in [17], we see a steady growth of the reduction with increasing search depth in LOA. Surprisingly, the increment of 1 is generating for the various depths a search tree that is only slightly larger than other increments. Apparently, the depth of the move explored is not so important in the history heuristic. Hence we may conclude that, unlike some data so far available for chess [16], the choice of the increment is of little value in the test environment we studied.

In the second series of experiments we looked at various increment parameter settings of the history table and the butterfly board (h,b) in our engine using the relative history heuristic and using a search depth of 14 ply. In Table 2 the total number of nodes searched for each combination of parameters is given. In the process of parameter tuning we found that (d^2, 2^d) is the most efficient. However, the difference with several other parameter configurations is not significant (e.g., (1,1), (d^2, d) or (2^d,1)). The difference between the best and worst parameter setting is 6 per cent in nodes searched. Hence, we may conclude that the exact choice of parameters seems to be not very critical.

5.2 Performance in LOA

In the third series of experiments we tested the added value of the relative history heuristic, using the optimal parameter setting of the previous subsection, against the same set of 171 LOA positions for several depths in our original search engine under different conditions.

Table 2. Performance of the relative history heuristic with different increments on a test set of 171 positions.

History Increment	Butterfly Increment	Total Nodes
1	1	12,261,241,807
1	d	12,544,923,748
1	d^2	12,936,458,114
1	2^d	12,741,580,747
d	1	12,654,462,037
d	d	12,433,892,630
d	d^2	12,914,014,566
d	2^d	13,075,535,903
d^2	1	12,501,473,310
d^2	d	12,238,059,952
d^2	d^2	12,509,830,417
d^2	2^d	12,234,575,562
2^d	1	12,354,762,028
2^d	d	13,081,065,785
2^d	d^2	12,928,253,841
2^d	2^d	12,954,976,931

In Figure 2 we plot the relative performance of the two heuristics defined as the size of the search tree investigated using the relative history heuristic divided by that of the standard history heuristic, as a function of the search depth. We observe that until depth 8 there is no significant difference between the two move-ordering schemes. From depth 9 onwards the difference increases with the depth to some 12 per cent at depth 14. We see that the search enhanced with the relative history heuristic searches fewer nodes than the one enhanced with the standard setting.

Since we have tuned our heuristic with this particular test set (Subsection 5.1), we performed a fourth series of experiments on a set consisting of 156 different positions[2] to validate the result. The positions were searched up to depth 15. In Figure 3 we see the relative performance of the two heuristics on the validation set. If we compare Figure 2 with Figure 3, we see that similar results are achieved.

Since the performance of many search enhancements may to some extent depend on the search engine, we modified the search engine in the fifth series of experiments by switching the multi-cut forward-pruning mechanism off. Because of the diminished forward pruning the sizes of our search trees increased and we were not able to conduct experiments at depth 13 and further. Looking at Figure 4 we see that the relative history heuristic decreases the search with 11 per cent at depth 12. Hence, we may conclude that the same pattern as in the previous

[2] The test set can be found at http://www.cs.unimaas.nl/m.winands/loa/VMP.zip.

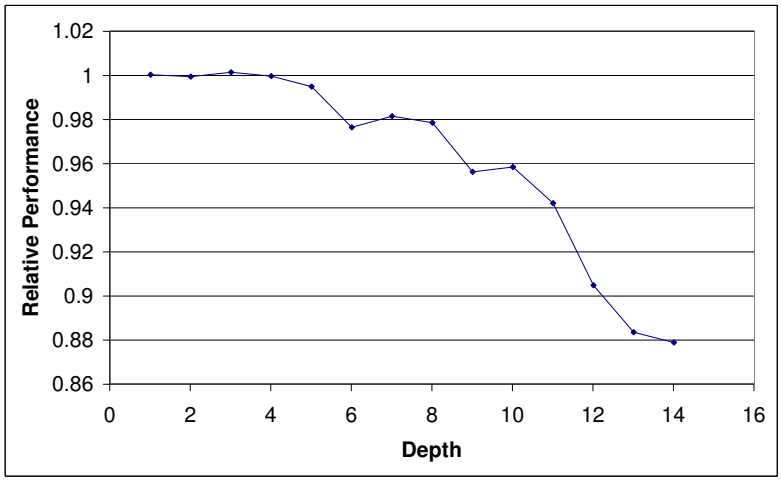

Fig. 2. Performance of the Relative History Heuristic.

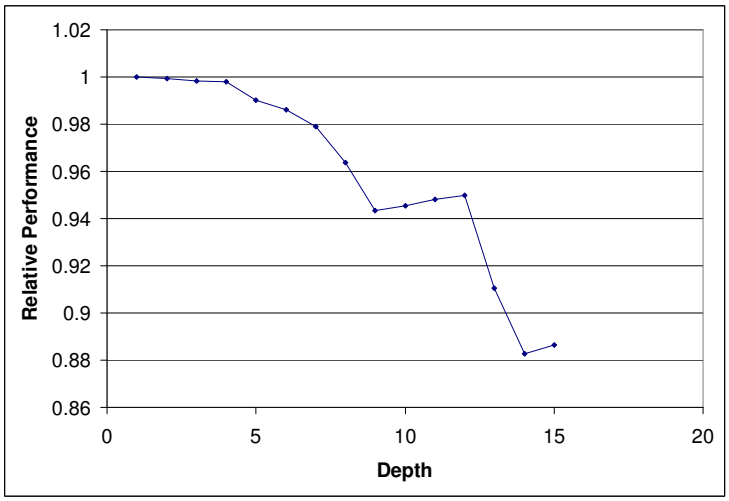

Fig. 3. Validating the Relative History Heuristic.

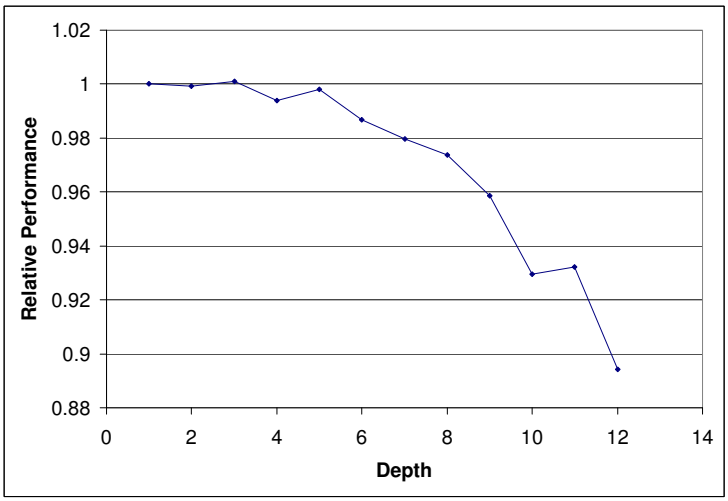

Fig. 4. Relative History Heuristic without using multi-cut.

experiments has started. We expect that this pattern will continue, which is to be confirmed in the future by more powerful machines.

5.3 Performance in Go

The relative history heuristic was designed for LOA. To investigate whether the relative history heuristic would be interesting for other domains too, we tested its performance on the small-board games of Go in the sixth series of experiments, for which we used the program MIGOS that recently had solved Go on the 5×5 board [20].

MIGOS uses an iterative-deepening PVS with a transposition table with 2^{24} double entries (using the *two-deep* replacement scheme), enhanced transposition cut-offs, symmetry lookups in the transposition table, internal unconditional bounds, and an enhanced move ordering in which the history heuristic is an important component. The implementation of the history heuristic employs one shared table for both the black and white moves which exploits the game-dependent property in Go that moves on the same intersection are often good for both sides. After some parameter tuning for the relative history heuristic increments, which we optimised for solving the empty 5×5 board, we found that using d^3 for both the history and the butterfly board gave quite promising results[3].

The current challenge in small-board Go is solving the 6×6 board (5×5 is the largest square board solved by a computer). Therefore we decided to test the performance of the relative history heuristic on a set of 24 problems for the 6×6

[3] We tested this combination on the LOA test set, too. Our experiments showed that this combination belongs to the better ones.

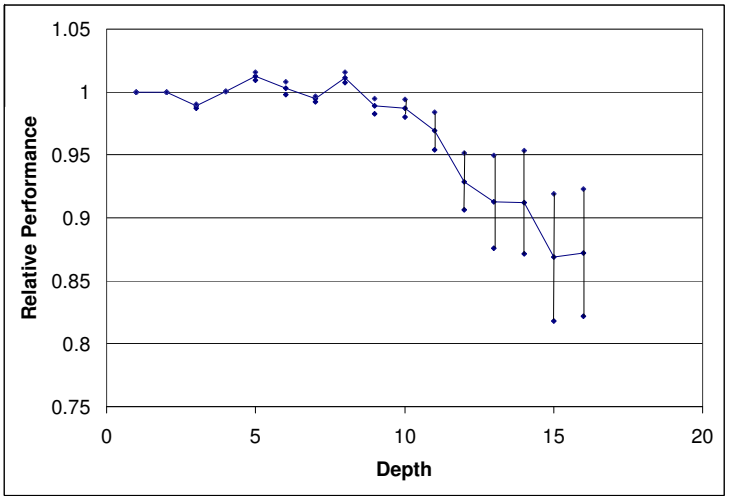

Fig. 5. Performance of the Relative History Heuristic in 6×6 Go.

board published in *Go World* by James Davies [5, 6]. Figure 5 shows the average relative performance of the relative history heuristic compared to the standard settings without a butterfly table. Since we only used a small number of test positions we also plotted the standard deviations. They tend to increase with the search depth. The reasons for this are (1) the exponential effect of changes in the move ordering, and (2) a reduction in the number of positions because some positions are already solved at smaller depths. The results indicate again that for shallow searches not much should be expected of using the relative history heuristic. However, after about 10 ply the first improvements become noticeable and at about 15 ply the relative history heuristic achieves a reduction of roughly 13 per cent. However, we remark that the test set is too small to draw strong conclusions. So far the results are favourable for the relative history heuristic and they indicate that the relative history heuristic is worth investigating in other domains as well.

6 Conclusion and Future Research

Combining the ideas of the history heuristic and the butterfly heuristic resulted in the relative history heuristic. This heuristic does not suffer from underestimating less frequently occurring moves in the search tree as the history heuristic does. We favour moves which are the good moves on average instead of moves which are the best move in absolute terms. Both the history heuristic and the relative history heuristic show a steady growth of the reduction with increasing search depth. Using the relative history heuristic our LOA program MIA searches even between 10 and 15 per cent fewer nodes (see Subsection 5.2). The

results were confirmed by the Go program MIGOS. Hence, we may conclude that the relative history heuristic is a valuable technique to order the moves in a game tree of considerable depth (more than 12 plies).

It is remarkable that the utility of increments other than 1 does not show much better performance in the (relative) history heuristic for our LOA program MIA. The good performance of the increment of 1 could be the result of some domain-dependent properties.

Finally, it would be interesting for future research to test our heuristic in still more different games, especially in Chess, since the original history heuristic was developed for Chess.

Acknowledgements We gratefully acknowledge financial support by the Universiteitsfonds Limburg / SWOL.

References

1. S.G. Akl and M.M. Newborn. The principal continuation and the killer heuristic. In *1977 ACM Annual Conference Proceedings*, pages 466–473. ACM, Seattle, USA, 1977.
2. Y. Björnsson and T.A. Marsland. Multi-cut alpha-beta pruning. In H.J. van den Herik and H. Iida, editors, *Computers and Games, Lecture Notes in Computing Science 1558*, pages 15–24. Springer-Verlag, Berlin, Germany, 1999.
3. D.M. Breuker, J.W.H.M. Uiterwijk, and H.J. van den Herik. Replacement schemes and two-level tables. *ICCA Journal*, 19(3):175–180, 1996.
4. M. Campbell, A.J. Hoane Jr., and F.-h. Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.
5. J. Davies. Small-board problems. *Go World*, 14-16:55–56, 1979.
6. J. Davies. Go in lilliput. *Go World*, 17:55–56, 1980.
7. C. Donninger. Null move and deep search: Selective-search heuristics for obtuse chess programs. *ICCA Journal*, 16(3):137–143, 1993.
8. D. Hartmann. Butterfly boards. *ICCA Journal*, 11(2-3):64–71, 1988.
9. E.A. Heinz. Adaptive null-move pruning. *ICCA Journal*, 22(3):123–132, 1999.
10. D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.
11. L. Kocsis. *Learning Search Decisions*. PhD thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2003.
12. L. Kocsis, J.W.H.M. Uiterwijk, and H.J. van den Herik. Move ordering using neural networks. In L. Montosori, J. Váncza, and M. Ali, editors, *Engineering of Intelligent Systems, Lecture Notes in Artificial Intelligence 2070*, pages 45–50. Springer-Verlag, Berlin, Germany, 2001.
13. T.A. Marsland and M. Campbell. Parallel search of strongly ordered game trees. *Computing Surveys*, 14(4):533–551, 1982.
14. S. Sackson. *A Gamut of Games*. Random House, New York, NY, USA, 1969.
15. J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.
16. J. Schaeffer. *Experiments in Search and Knowledge*. PhD thesis, Department of Computing Science, University of Waterloo, Canada, 1986.

17. J. Schaeffer. The history heuristic and the performance of alpha-beta enhancements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.
18. J. Schaeffer and A. Plaat. New advances in alpha-beta searching. In *Proceedings of the 1996 ACM 24th Annual Conference on Computer Science*, pages 124–130. ACM Press, New York, NY, USA, 1996.
19. Y. Tsuruoka, D. Yokoyama, and T. Chikayama. Game-tree search algorithm based on realization probability. *ICGA Journal*, 25(3):132–144, 2002.
20. E.C.D. van der Werf, H.J. van den Herik, and J.W.H.M. Uiterwijk. Solving Go on small boards. *ICGA Journal*, 26(2):92–107, 2003.
21. M.H.M. Winands, J.W.H.M. Uiterwijk, and H.J. van den Herik. The quad heuristic in Lines of Action. *ICGA Journal*, 24(1):3–15, 2001.
22. M.H.M. Winands, H.J. van den Herik, J.W.H.M. Uiterwijk, and E.C.D. van der Werf. Enhanced forward pruning. *Information Sciences*, 2004. Accepted for publication.