Combining Proof-Number Search with Alpha-Beta Search

Mark H.M. Winands Jos W.H.M. Uiterwijk
H. Jaap van den Herik

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

Abstract

In this paper we present a method of combining proof-number (PN) search with $\alpha\beta$ search. We focus on real-time applications in the endgame of Lines of Action. Nodes once proven by PN search are stored in a transposition table (PN-TT), which is used in the $\alpha\beta$ search. Our experiments reveal that this method is successful in the domain of Lines of Action. The use of the PN-TT is necessary, since a combination of plain PN search with $\alpha\beta$ search is not successful.

1 Introduction

Most modern game-playing computer programs successfully use an $\alpha\beta$ search with enhancements for *online* game-playing. Unfortunately, the $\alpha\beta$ search per se sometimes is not sufficient to play well in the endgame. In some games, such as chess, this problem can be solved by the use of endgame databases [11] in the $\alpha\beta$ search. One of the drawbacks of this method is that it is only feasible, due to memory constraints, for endgames with a relatively small state-space complexity. An alternative approach is the use of a specialised binary (win or non-win) search method, like proof-number (PN) search [4]. In some games PN search outperforms $\alpha\beta$ search in proving the game-theoretic value of endgame positions. PN search has been applied successfully *offline* to the endgame of Awari [4], chess [7], checkers [14] and Shogi [15]. In this paper we introduce a method of combining PN search and $\alpha\beta$ search *online* instead of *offline*. Also the benefit of using a transposition table to reuse some knowledge achieved during the PN search, in the $\alpha\beta$ search is examined. We remark that our method of combining perfect and heuristic evaluations is much simpler to apply than Beal's nested Minimax [5], using heuristic values together with perfect upper and lower bounds.

The remainder of this paper is organised as follows. Section 2 gives a short description of the PN-search algorithm. Moreover, the implementation and a method of using a transposition table in PN search are given. The combination of PN and $\alpha\beta$, called PN-$\alpha\beta$, is described in Section 3. The game of Lines of Action

(LOA) and the experimental set-up are explained in Section 4. Subsequently, the results of using PN-$\alpha\beta$ in LOA are presented in Section 5. Finally, in Section 6 we present our conclusions and propose topics for future research.

2 Description of Proof-Number search

Proof-number (PN) search is a best-first search algorithm especially suited for finding the game-theoretical value in game trees [2]. Its aim is to prove the true value of the root of the tree. A tree can have three values: *true*, *false* or *unknown*. In the case of a forced win, the tree is *proved* and its value is true. In the case of a forced loss or draw, the tree is *disproved* and its value is false. Otherwise the value of the tree is unknown. In contrast to other best-first algorithms PN search does not need a domain-dependent heuristic evaluation function in order to determine the most-promising node (to be expanded next) [7]. PN search selects the most-promising node using two criteria: (1) the shape of the search tree (the number of children of every internal node) and (2) the values of the leaves. These two criteria enable PN search to treat efficiently game trees with a non-uniform branching factor. A disadvantage of PN search is that the whole search tree has to be stored in memory. When the memory is full, the search process has to be terminated prematurely or countermeasures have to be taken [2, 6].

2.1 Transposition tables in PN search

Since we would like to reuse some of the information obtained in the PN search, a transposition table is used. Ideally, we would store every (proved) position encountered in the search tree, but unfortunately this is not possible due to memory constraints. Therefore, a proved-node transposition table (PN-TT) is implemented as a hash table. The well-known Zobrist-hashing method [18] is used.

Each entry in the transposition table has a length of 64 bits. These bits are reserved for the hash key to distinguish among different positions having the same hash index. When two positions have the same hash index, the last examined position is preferred over early ones (replacement scheme *New*, see [9]). The usual schemes *Big* or *Deep* are not used, because the number of nodes of a subtree or the depth of a subtree are not appropriate measures for replacement in case of PN search. A position is stored if and only if it is a proved interior node. At present, disproved nodes are not stored because it is not known whether the game-theoretic value of the node is a loss or a draw. If it is known that draws are impossible in a game (such as Hex), then it is possible to store also the disproved nodes in a separate table and to use them in an analogous way as proved nodes in $\alpha\beta$ (see Section 3).

The transposition table filled by PN search can be used in the $\alpha\beta$ search (of course, it is also used during the PN search itself). The procedure is as follows. Before the expansion of the selected best node the position is looked up in the transposition table. If it is not found, the node is expanded by generating its children one by one. Looking up a position is done after checking on repetitions but before checking on terminal positions. Preliminary experiments using a set of 61 proven positions showed that PN search with PN-TT evaluates 10% fewer nodes than PN search without PN-TT. An exhaustive examination of the use of the PN-TT in PN search is beyond the scope of this paper.

3 PN-$\alpha\beta$

The idea of combining PN search and $\alpha\beta$ search originates from Van der Meulen who used PN search in combination with $\alpha\beta$ search [2] in the Awari program LITHIDION [3]. In the opponent's time LITHIDION performed PN searches on its potential moves looking for wins. When the opponent selected a losing move, LITHIDION used the outcome of the PN search to select the winning move. The weakness of the method is that is impossible to perform deep PN searches for *all* possible opponent moves in the opponent's time when the number of moves is high.

In this article, we introduce another way to combine the two search methods consecutively. We denote this combination by PN-$\alpha\beta$. PN is done in the player's own time and information gained in the PN search is used in the $\alpha\beta$ search. PN-$\alpha\beta$ works as follows. In endgame positions PN search is applied for some fixed fraction of the allotted time for a move. Here, an endgame position is defined as a position occurring in a game after a fixed number of moves. The PN search terminates prematurely when either the position is proved or disproved, or the system runs out of memory. When the position can be proved, a winning move is played. It is possible that this move is not optimal (i.e., it will not lead to a shortest win). If the PN-search algorithm is not able to find a winning move, the $\alpha\beta$-search algorithm is applied for the remaining time. During the PN search the transposition table is used as described in Subsection 2.1. Whenever a position is examined in the $\alpha\beta$ search, the position is looked up in the PN-TT for a possible cut-off. Between the moves the transposition table is not cleared because the stored positions may still be useful during the game. The number of moves defining an endgame position and the fraction of time allotted to PN search are controllable parameters, to be fine-tuned.

4 Experimental setup

In this section we briefly explain the game of Lines of Action, which is used as the test domain. Then we provide some details on the experiments performed.

4.1 Lines of Action

Lines of Action (LOA) [12] 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 (see Figure 1), 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 (see Figure 2). 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 into one unit (see Figure 3). The connections within the unit may be either orthogonal or diagonal. If a player cannot move, this player loses. If a position with the same player to move occurs for the third time, the game is drawn.

Figure 1: The initial position of LOA.

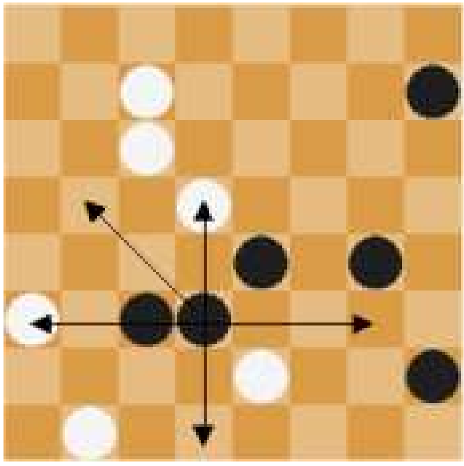

Figure 2: An example of possible moves in a LOA game.

LOA has an average game length of 38 plies and an average branching factor of 30 [16]. An interesting property of the game is that most terminal positions still have more than 10 pieces remaining on the board, which makes the game not suitable for endgame databases. Although reasonable effort has been undertaken to construct adequate evaluation functions for LOA [17], experiments still suggest that these are not very good predictions in the case of forced wins. Therefore, LOA seems an appropriate test domain for PN-$\alpha\beta$ search.

4.2 Experimental details

All experiments have been performed in the framework of the tournament program MIA (Maastricht In Action). MIA[1] has been written in Java and runs on every well-known operating system. MIA performs an $\alpha\beta$ depth-first iterative-deepening search. The program uses a *two-deep* transposition table [8], the history heuristic [13] and killer moves [1]. For all experiments described in this paper, the null move is not used because of possible negative effects of zugzwang in LOA [16]. In the leaf nodes of the tree a quiescence search is performed. This quiescence search

[1]It can be played at the website: http://www.cs.unimaas.nl/m.winands/loa/.

Figure 3: A terminal LOA position.

looks at capture moves, which form or destroy connections [17]. The evaluation function used in MIA is the so-called quad evaluator, which is based on the *quad heuristic* [17].

After some experiments we found that the best strategy to enable the PN-$\alpha\beta$ in MIA was to exploit it in positions after 17 moves (34 plies) using a quarter of the time allotted for a move. Since the time was limited to 30 seconds per move, it means that PN search is trying to solve the position in 7.5 seconds when to move. In that time the memory will not completely be filled. Regarding the experiments we only take into account games which last longer than 34 plies. Drawn games are also not taken into account. Five series of experiments have been played. The first experiment estimated the proportion of wins by Black and White when both sides use plain $\alpha\beta$ search in a series of 300 games. This proportion is taken as a reference for the other experiments. The second and third experiment measured the advantage of PN-$\alpha\beta$ over plain $\alpha\beta$ for each colour. The fourth and fifth experiment measured the advantage of PN-$\alpha\beta$ without using PN-TT over plain $\alpha\beta$ for each side.

5 Experimental Results

Table 1 provides the results for the five series of experiments. The second and third column give the absolute and relative game scores, respectively. The table shows that PN-$\alpha\beta$ search has an advantage over normal $\alpha\beta$ search, but only when information gathered in the PN search is reused by the transposition table. If we take the reference ratio of Black and White win percentages into account, we see that enabling PN-$\alpha\beta$ yields an increase in performance of some 6%, irrespective of colour.

In these experiments using PN search even has a negative effect when the infor-

Black	White	Absolute	Relative
$\alpha\beta$	$\alpha\beta$	159-131	55-45
PN-$\alpha\beta$ with PN-TT	$\alpha\beta$	122-78	61-39
$\alpha\beta$	PN-$\alpha\beta$ with PN-TT	96-104	48-52
PN-$\alpha\beta$ without PN-TT	$\alpha\beta$	150-149	50-50
$\alpha\beta$	PN-$\alpha\beta$ without PN-TT	104-81	57-43

Table 1: Experimental results

mation is not reused. A potential reason is the following. If we have performed an unsuccessful PN search, $\alpha\beta$ can search less deep (approximately one ply) because of the time used. The PN-TT is making up for this in two ways. First, when a position occurs in the PN-TT it has not to be explored or evaluated anymore. Despite of the time used for doing the PN search, it can even happen that $\alpha\beta$ can search more deeply than a normal search due to the PN-TT. For instance, in the position of Figure 4, PN search was not able to solve this position in 7.5 seconds, but due to the PN-TT the $\alpha\beta$ search reached a depth of 11 plies. Using plain $\alpha\beta$ the search only reached a depth of 9 plies. Second, the heuristic error of the evaluation function will be minor, which can result in other (better) move decisions.

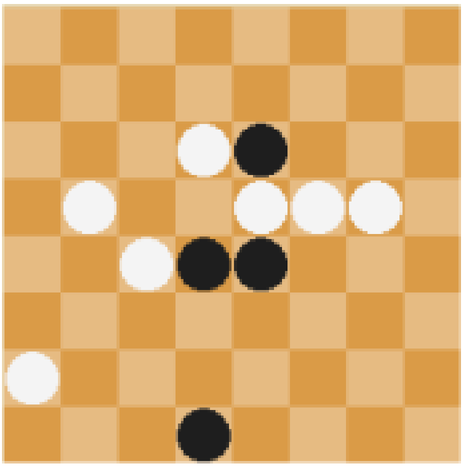

Figure 4: Black to move and to win in 21 plies. MONA vs. YL, Game 4, http://www.cs.ualberta.ca/~darse/LOA/, March 2000.

Although the method is quite successful, we still have not solved the problem when to use PN search. Using PN search after a fixed number of moves is an artificial solution. Clearly, for each different opponent, these parameters have to be determined separately. Nevertheless, even then PN search will sometimes still be activated too late (or too early). In Table 2 we have selected from Table 1 only the games with over 45 plies. Although the set is too small for meaningful conclusions, it seems that unsuccessfully applying PN search for too long has a negative effect. In other words, sometimes it is better to invalidate PN-$\alpha\beta$ in the very end of a game. In conclusion, it remains a challenge to find a *dynamic* strategy, which determines on a per-move basis when to apply PN search in a game. We believe that a good heuristic may improve the results of Table 1 considerably.

Black	White	absolute	relative
$\alpha\beta$	$\alpha\beta$	22-18	55-45
PN-$\alpha\beta$ with PN-TT	$\alpha\beta$	21-21	50-50
$\alpha\beta$	PN-$\alpha\beta$ with PN-TT	21-18	54-46
PN-$\alpha\beta$ without PN-TT	$\alpha\beta$	38-36	51-49
$\alpha\beta$	PN-$\alpha\beta$ without PN-TT	31-17	65-35

Table 2: Experimental results for games with over 45 plies.

An important issue is that the framework of MIA is appropriate for the beneficial use of a PN-TT. According to Breuker *et al.* [6] PN search explores some positions more deeply than an $\alpha\beta$ search would do. The information gained in those deep searches is not useful in a narrow $\alpha\beta$ search, but MIA uses a search extension in the form of a quiescence search [17], which can lead to deeper paths than usual in depth-limited $\alpha\beta$ search. In this sense MIA is somewhat biased in the favour of this method of using information from the PN search.

6 Conclusions and future research

We conclude that our method of combining PN search and $\alpha\beta$ outperforms plain $\alpha\beta$ search in the tournament program MIA. We note that the kind of games played by MIA potentially results in PN-search friendly positions, i.e., positions with many forced moves. Therefore, it has to be tested whether this approach is also profitable in other LOA programs. It might happen that PN-$\alpha\beta$ is compensating for the possible weak play of MIA in the endgame. Another conclusion is that the proved-nodes transposition table makes up for the loss of time when the position is not proved.

Moreover, we have identified two problems which are not solved satisfactorily. The first problem is that we do not use the information of the disproved nodes. The second problem is that there is no dynamic strategy available which determines when to use PN-$\alpha\beta$ search instead of $\alpha\beta$. These two problems will be subject of future research.

Finally, we would like to mention that the graph-history-interaction problem (cf. [10]), especially the GHI evaluation problem, can occur in LOA too. For instance, draws can be agreed upon due to the three-fold-repetition rule. However, dependent on its history a node can be a draw or have another value. Therefore, it is possible that due to the use of the PN transposition table, nodes are given a wrong value. More detailed experiments have to be performed to reveal if this will be a serious disadvantage. In that case a solution like proposed in [10] will be implemented.

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, 1977.

[2] L.V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. Thesis, University of Limburg, The Netherlands, 1994.

[3] L.V. Allis, M. van der Meulen, and H.J. van den Herik. Databases in Awari. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2: the second computer olympiad*, pages 73–86. Ellis Horwood, Chichester, 1991.

[4] L.V. Allis, M. van der Meulen, and H.J. van den Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–123, 1994.

[5] D.F. Beal. Mixing heuristic and perfect evaluations: Nested minimax. *ICCA Journal*, 7(1):10–15, 1984.

[6] D.M. Breuker. *Memory versus Search in Games*. Ph.D. Thesis, Universiteit Maastricht, The Netherlands, 1998.

[7] D.M. Breuker, L.V. Allis, and H.J. van den Herik. How to mate: Applying proof-number search. In H.J. van den Herik, I.S. Herschberg, and J.W.H.M. Uiterwijk, editors, *Advances in Computer Chess 7*, pages 251–272. University of Limburg, Maastricht, The Netherlands, 1994.

[8] 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.

[9] D.M. Breuker, H.J. van den Herik, J.W.H.M. Uiterwijk, and L.V. Allis. Replacement schemes for transposition tables. *ICCA Journal*, 14(4):183–193, 1994.

[10] D.M. Breuker, H.J. van den Herik, J.W.H.M. Uiterwijk, and L.V. Allis. A solution to the GHI problem for best-first search. *Theoretical Computer Science*, 252(1-2):121–149, 2001.

[11] H.J. van den Herik and I.S. Herschberg. The construction of an omniscient endgame data base. *ICCA Journal*, 8(2):66–87, 1985.

[12] S. Sackson. *A Gamut of Games*. Random House, New York, NY, USA, 1969.

[13] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[14] J. Schaeffer and R. Lake. Solving the game of checkers. In J. Nowakowski, editor, *Games of No Chance*, pages 119–133. Cambridge University Press, Cambridge, UK, 1996.

[15] M. Seo, H. Iida, and J.W.H.M. Uiterwijk. The PN*-search algorithm: Application to tsume-shogi. *Artificial Intelligence*, 129(1-2):253–277, 2001.

[16] M.H.M. Winands. *Analysis and Implementation of Lines of Action*. M.Sc. Thesis, Universiteit Maastricht, The Netherlands, 2000.

[17] 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.

[18] A.L. Zobrist. A new hashing method for game playing. Technical Report 88, Computer Science Department, The University of Wisconsin, Madison, WI, USA, 1970. Reprinted (1990) in *ICCA Journal*, 13(2):69–73.