AN EVALUATION FUNCTION FOR LINES OF ACTION

M.H.M. Winands, H.J. van den Herik, J.W.H.M. Uiterwijk
Institute for Knowledge and Agent Technology, Department of Computer Science, Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht, The Netherlands
{m.winands,herik,uiterwijk}@cs.unimaas.nl, http://www.cs.unimaas.nl/m.winands/

Abstract Lines of Action (LOA) is a two-person zero-sum chess-like connection game. Building an evaluation function for LOA is a difficult task because not much knowledge about the game is available. In this paper the evaluation function of the tournament program MIA is explained. This evaluator consists of the following nine features: concentration, centralisation, centre-of-mass position, quads, mobility, walls, connectedness, uniformity, and player to move. These features have resulted in the evaluator MIA IV. The evaluator is tested in a tournament against other LOA evaluators, which have performed well at the previous Computer Olympiads. Experiments show that MIA IV defeats them with large margins. It turns out that the evaluator even performs better at deeper searches.

Keywords: Lines of Action, evaluation function, MIA

1. Introduction

LOA is a two-person zero-sum game with perfect information; it is a chess-like game with a connection-based goal, played on an 8×8 board. LOA was invented by Claude Soucie around 1960. Sid Sackson (1969) described it in his first edition of *A Gamut of Games*. After this publication, LOA received some attention of AI researchers. For instance, the first LOA program was written at the Stanford AI laboratory around 1975 by an unknown author. In the 1980s and 1990s "hobby" programmers wrote several LOA programs. However, all were beatable by humans (Dyer, 2000). At the end of the nineties LOA again became a target of AI researchers. Some of them used LOA only as a test domain for their algorithms, others tried to build strong LOA programs by using new ideas. The programs YL, MONA and MIA (Maastricht In Action) belong to the latter category. MIA finished third, second and again second at the fifth, sixth and seventh Computer Olympiad, respectively (Björnsson, 2000; Björnsson and Winands, 2001; Björnsson and Winands, 2002). The program can be played online at the website: http://www.cs.unimaas.nl/m.winands/loa/.

The standard framework of the $\alpha\beta$ search with its enhancements offers a good start for building a strong game-playing program. The real challenge in LOA is building a decent evaluation function, which incorporates the strategic intricacies of the game. The difficulty lies in the fact that knowledge about LOA evaluation functions is not well developed, although some material on this topic has been published (Winands et al., 2001). In this paper we discuss the latest evaluation function used in the program MIA.

The remainder of this paper is organised as follows. Section 2 explains the game of Lines of Action and describes the search engine. In Section 3 the evaluation function is explained. This evaluation function is tested against other evaluators in Section 4. Finally, in Section 5 we present our conclusions and topics for future research.

2. Test Environment

In this section we explain first the game of Lines of Action. Next, the search engine of MIA is described briefly.

2.1 Lines of Action

LOA is played on an 8×8 board by two sides, Black and White. Each side has twelve pieces at its disposal. 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. The connections within the unit may be either orthogonal or diagonal. In the case of simultaneous connection, the game is drawn. 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.

Analysis of 2585 self-play matches showed an average branching factor of 29 and an average game length of 44 ply. The game-tree complexity and state-space complexity are estimated to be $O(10^{23})$ (Winands et al., 2001) and $O(10^{64})$, respectively. A characteristic property of LOA is that it is a converging game (Allis, 1994), since the initial position consists of 24 pieces, and during the game the number of pieces (usually) decreases. However, since most terminal positions have still more than 10 pieces remaining on the board (Winands, 2000), endgame databases are (probably) not effectively applicable in LOA. As a case in point, we remark that an endgame database of ten pieces would require approximately 10 terabytes. Finally, in LOA the standard chess notation for moves is used.

2.2 MIA's Search Engine

MIA 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 (Marsland and Campbell, 1982). A *two-deep* transposition table (Breuker et al., 1996) 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 (Schaeffer and Plaat, 1996). Next, a null move (Donninger, 1993) 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 (Knuth and Moore, 1975; Marsland and Campbell, 1982). The null move is done at CUT nodes *and* at ALL nodes. At a CUT node a variable scheme, called adaptive null move (Heinz, 1999), 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 for setting *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, multi-cut (Björnsson and Marsland, 1999) is performed. Experiments showed that using multi-cut is not only beneficial at CUT nodes but also at ALL nodes (Winands et al., 2003). For move ordering, the move stored in the transposition table, if applicable, is always tried first. Next, two killer moves (Akl and Newborn, 1977) 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 history table (Schaeffer, 1983). In the leaf nodes of the tree a quiescence search is performed. This quiescence search looks at capture moves, which form or destroy connections (Winands et al., 2001) and at capture moves going to the central 4×4 board.

3. Evaluation Function

In this section the evaluation function of MIA is explained. This evaluator consists of the following nine features: *concentration*, *centralisation*, *centre-of-mass position*, *quads*, *mobility*, *walls*, *connectedness*, *uniformity*, and *player to move*. These features are described below in detail (Subsection 3.1 to 3.9), followed by some information about the use of caching (Subsection 3.10).

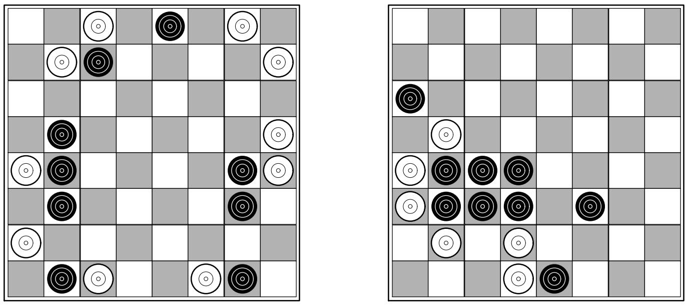


Figure 1. (a) Scattered Pieces (b) Position with two black Q_4's.

3.1 Concentration

The concentration of the pieces is calculated by a centre-of-mass approach. In MIA this is done in four steps. First, the centre of mass of the pieces on the board is computed for each side. Second, we compute for each piece its distance to the centre of mass. The distance is measured as the minimal number of squares from the piece to the centre of mass. These distances are summed together, called the sum-of-distances. Third, the sum-of-minimal-distances is looked up in a table. It is defined as the sum of the minimal distances of the pieces from the centre of mass. This number is necessary since otherwise boards with a few pieces would be preferred. For instance, if we have ten pieces, there will be always eight pieces at a distance of at least 1 from the centre of mass, and one piece at a distance of at least 2. In this case the sum-of-minimal-distances is 10. Thus, the sum-of-minimal-distances is subtracted from the sum-of-distances, the result being called the surplus-of-distances. Fourth, we calculate the concentration, defined as the inverse of the average surplus-of-distances. Since by doing so we reward positions with pieces in the neighbourhood of each other, eventually they will be connected in solid formations or they will create threats to win.

3.2 Centralisation

Each piece gets a value dependent on its board square according to this feature. Pieces at squares closer to the centre are given higher values than the ones farther away. Pieces at the edge are given a negative value. This is done because such pieces are easy to block by a wall (see Subsection 3.6). Pieces at the corner are punished even more severely. To prevent the program from over-aggressively capturing pieces, the average is computed instead of the sum of piece values.

3.3 Centre-of-mass Position

In earlier versions of MIA positions with a somewhat more centralised centre-of-mass were slightly preferred. The idea was to prevent formations from being built on the edges, where they are more easily destroyed or blocked. Interestingly, after applying Temporal-Difference (TD) learning the weight for the centralised centre-of-mass feature is changing its sign (Winands et al., 2002), which means that opposite to expectations it is good to have the centre-of-mass closer to the edge instead of in the centre. If the centre-of-mass is in the centre, it is possible that pieces are scattered over the board (e.g., the white pieces in Figure 1a). If the centre of mass is at the edge, pieces have to be in the neighbourhood of each other, otherwise they would lie outside the board. Another plausible explanation of why it is worse to have the main piece formation in the centre is that it can be more easily attacked there, whereas groups residing closer to the edge can only be attacked from one side.

3.4 Quads

The use of quads for a LOA evaluation function was first proposed and implemented by Dave Dyer in 1996 in his program LOAJAVA and empirically evaluated by Winands et al. (2001) . This feature counts certain quads types. A quad is defined as a 2×2 array of squares (Gray, 1971). In this feature we only consider quads of three (Q_3) or four pieces (Q_4) of the same colour, since it is impossible to destroy these formations by a single capture. However, the danger exists that many of those quads are created outside the neighbourhood of the centre of mass. So, in MIA we have rewarded only Q_3's and Q_4's, which are at a distance of at most two of the centre of mass. For instance, Black has two Q_4's in Figure 1b.

3.5 Mobility

In the mobility feature the number of moves for each side are computed. This feature was first implemented in MONA and YL. In previous evaluation functions of MIA all moves were weighted equally. However, experiments have shown that certain move types are better than others (see also Hashimoto et al., 2003). Therefore, in MIA the following bonus/malus system is applied: the value of a capture move is doubled, the value of a move going to an edge or a move along an edge is halved. If a move belongs to multiple categories, the bonus/malus system is used multiple times. For example, let us assume that a regular move gets value 1, then a capture move gets value 2, a capture move going to an edge gets value 1, a capture move in an edge line going to a corner gets value 0.5. The computational requirements of this component are not high. For each line configuration of pieces (represented as a bit vector) the mobility can be precomputed and stored in a table. During the search, the index scheme can be updated incrementally and in the evaluation function only a few table lookups have to be done.

3.6 Walls

Because a piece is not allowed to jump over the opponent's pieces, it can happen that the piece is blocked, i.e., cannot move. Blocking a piece far away from the other pieces is an effective way of preventing the opponent to win. Even partial blocking, called a wall (Handscomb, 2000), can be quite effective, since it forces a player to find a way around the wall. Detecting whether a piece is (partially) blocked can be expensive as we have to know what the moves of the piece are and what its goal is. In MIA we look only at walls that prevent the opponent's edge pieces from moving toward the centre. These walls are quite common and effective. The patterns can be precomputed and therefore are easy to detect. For example, in Figure 2a the piece on **a4** is blocked in three ways going to the centre, whereas the piece on **h4** is only blocked in two centre directions. In the evaluator, we distinguish between walls which block two or three centre directions. We also remark that we take special care of walls which block corner pieces. For example, the piece on **h8** is blocked only in two directions, but we evaluate this position as if it was blocked in 3 centre directions. The totally isolated piece on **a8** is evaluated as if there were two walls which both block the piece in three directions. We only look at certain blocking patterns for edge pieces. For example, the pieces on **b1** and **c1** are completely blocked, but we take only the two 3-centre directions blocks into account. It is a subject of future research to incorporate more of these kind of patterns.

3.7 Connectedness

Although the concentration component and quad component favour solid formations in the centre, there is still room for a component which determines the connectedness of a side. In MIA we compute the average number of connections of a piece. In some evaluation functions the total number of connections is taken into account (e.g., YL), but this could implicitly be a material advantage. Any kind of material component in LOA evaluation functions is always tricky because the program might wildly capture pieces. This feature does not take into account whether a connection is important or not. To distinguish this, a global look at the board would be needed, which is time consuming. The number of connections for each side in each line configuration can be precomputed as is done with the mobility component.

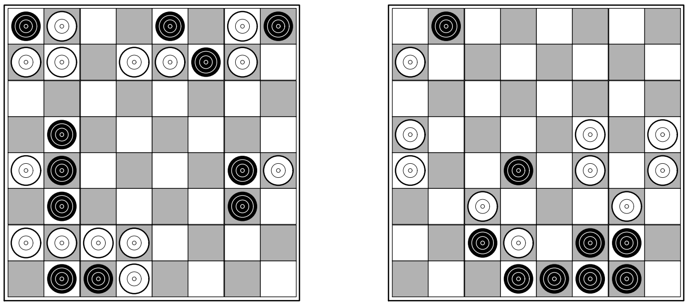


Figure 2. (a) Position with walls (b) Position with an outlier on **b8**.

3.8 Uniformity

The disadvantage of the centre-of-mass approach is that it aims to connect as many pieces as possible in a local group, hardly worrying about some remote pieces (orphans). It is sometimes hard to connect these orphans. For instance, in Figure 2b the black pieces are grouped around **e2**, but the black piece on **b8** is rather far away from this group. To prevent that one or more pieces become too remote from the main group, a feature is used which aims at a uniform distribution (Chaunier and Handscomb, 2001) to counterbalance the negative effects of the centre-of-mass approach. In our program this is done in a way which is primitive but effective. The area of the distributed pieces is computed, assuming it is a rectangle. The smaller the area is, the higher the reward is. An analogous implementation was first done in the program YL, but details are not known.

3.9 Player to Move

In the search tree not every leaf node has the same player to move. A small bonus is given to the moving side, since having the initiative is mostly an advantage in LOA (Winands, 2000) like in many other games (Uiterwijk and van den Herik, 2000).

3.10 Caching

It is possible in our evaluation function to cache computations of certain features, which can be used in other positions. Let us assume that we investigate the move **b8-c8** in Figure 2b and evaluate the resulting position. If we next investigate **b8-b7** we notice that certain properties of White's position remain the same (e.g., the number of pieces, centre-of-mass, the number of connections), whereas others can change (e.g., moves, blockades). It easy to see that we do not have to compute the concentration, centralisation, position of the centre-of-mass, quads, connection, and uniformity for White again. Evaluation of components, which are not dependent of the position of the other side, are stored in the evaluation cache table. In the current evaluation function this gives a speed-up of at least 60 percent in the number of nodes investigated per second.

4. Experiments

In order to quantify the improvements of the evaluation function, we played a round-robin tournament in which evaluators from earlier tournament versions of the program participated. All evaluators used the current search engine, described in Subsection 2.2. The evaluators are explained in Subsection 4.1. The results are described in Subsection 4.2.

4.1 Benchmark Evaluators

The benchmark evaluation functions are described below.

Evaluator: MIA I The core of this evaluation function is the centre-of-mass approach. The quad feature is also implemented. Pieces at the edge are given a negative bonus. Contrary to MIA IV a bonus is given for a centralised centre-of-mass (Winands et al., 2001). The weights of the features were carefully hand-tuned. In retrospect this evaluator was primitive, although it won a game against both MONA and YL at the fifth Computer Olympiad (Björnsson, 2000).

Evaluator: MIA II The major change of this evaluation function compared to the previous one is the introduction of the mobility component. There is no discrimination in rewarding different move types. In this evaluator pieces at a corner edge are punished more severely. Using this evaluator the tournament program shared the first place with YL in the regular tournament at the sixth Computer Olympiad. The play-off match was won by YL (Björnsson and Winands, 2001).

Evaluator: MIA III This evaluation function is enhanced with the wall feature. The centralisation feature is improved by rewarding pieces in the centre. A bonus is given for the player to move. The major improvement was retuning all the weights by using TD-learning (Winands et al., 2002). There were three major changes in the weights. First, the initial weight of the dominating centre-of-mass was decreased to one tenth of its original value, indicating that we had overestimated the importance of this feature. Second, the weight for the centralised centre-of-mass feature changed its sign, which means that opposite to expectations it is good to have the centre-of-mass closer to the edge instead of in the centre. Third, the weight of the centralisation component increased the most, indicating that we had overestimated the importance of this feature. Using this evaluator the tournament program finished second at the seventh Computer Olympiad (scoring 1.5 points out of 4 against the much improved winner YL) (Björnsson and Winands, 2002). An exhibition match was played against MONA during the *Third International Conference on Computers and Games 2002 (CG'02)*, which ended in a 2-2 tie (Billings and Björnsson, 2002).

Evaluator: MIA IV This evaluation function incorporates all features as described in Section 3. The centralisation, wall, and player-to-move features used the same weights as the ones in MIA III. All the weights of the other features were basically found by using TD-learning. Some of them were adjusted by hand afterwards.

An overview of the separate features as used in the four evaluators is given in Table 1. Note that the weights and details of the features may differ between different evaluators.

	MIA I	MIA II	MIA III	MIA IV
Concentration	X	X	X	X
Centralisation	X	X	X	X
C.o.m. position	X	X	X	X
Quads	X	X	X	X
Mobility		X	X	X
Walls			X	X
Connectedness				X
Uniformity				X
Player to move			X	X

Table 1. Overview of the features.

4.2 Results

The evaluators, previously described, played 1000 matches against each other in a round-robin tournament. They started always from the same 10 positions given in the Appendix, playing with both colours. To prevent that programs played the games over and over again, a sufficiently large random factor was included in each evaluation function.

Fixed-depth searches were used as time control instead of time. At first sight it may look as if we are favouring the more advanced evaluators (i.e., they are time intensive because of the extra knowledge). This is not a problem for two reasons. First, the difference in speed is quite moderate. The program runs only 15 per cent slower with the MIA IV evaluator than with the MIA I evaluator. All the evaluators have to compute the average distance to the centre-of-mass and the quads, which is time consuming. Most other additions are relatively cheap. Second, when an evaluator is a good predictor of the situation, a best move found at a shallow search is more likely to stay good and therefore causing cut-offs at deeper searches. For example, when the MIA I evaluator is used in the current search engine it searches 75 per cent more nodes compared to the

MIA IV evaluator. The advantage of fixing the depth is that we can measure the influence of increasing the depth.

Evaluator	MIA I	MIA II	MIA III	MIA IV
MIA I	0	259	199	71.5
MIA II	741	0	373	163.5
MIA III	801	627	0	248.5
MIA IV	928.5	836.5	751.5	0

Table 2. Tournament results at depth 4.

Evaluator	MIA I	MIA II	MIA III	MIA IV
MIA I	0	188	168.5	51
MIA II	812	0	356	174
MIA III	831.5	644	0	223.5
MIA IV	949	826	776.5	0

Table 3. Tournament results at depth 6.

Evaluator	MIA I	MIA II	MIA III	MIA IV
MIA I	0	137	159.5	41.5
MIA II	863	0	360	129
MIA III	840.5	640	0	205
MIA IV	958.5	871	795.0	0

Table 4. Tournament results at depth 8.

Evaluator	MIA I	MIA II	MIA III	MIA IV
MIA I	0	97.5	137.5	44.5
MIA II	902.5	0	359.5	121.5
MIA III	862.5	640.5	0	234.5
MIA IV	955.5	878.5	765.5	0

Table 5. Tournament results at depth 10.

In Tables 2-5 the results of the tournaments are given for searches to depth 4, 6, 8, and 10, respectively. MIA IV defeats the previous evaluators of MIA with ease. Even the strong MIA III is not able to score more than 20 to 25 percent of the points against MIA IV. Although MIA II's only major improvement is a primitive mobility component, it did not only outperform MIA I, but it also played much better against MIA III and IV than MIA I did. Interestingly, the weak MIA I performs worse at deep searches, whereas the opposite holds for the strong MIA IV evaluator. A reason might be that at the one hand a deep search is not able to compensate the lack of knowledge of MIA I, while at the other hand a deep search exploits more of the potential of MIA IV.

5. Conclusions and Future Research

In this paper we have seen that MIA IV defeats the older evaluators by large margins. Most additions of MIA IV in knowledge are quite simple to evaluate and lead to big rewards in playing strength. It turns out that MIA IV even performs better at deeper searches.

More patterns of blocked pieces, better distinction of move types in the mobility component, and additional knowledge whether a connection is important are some of the issues which could improve the evaluator. There is still room to fine tune certain weights and parameters in the evaluation function. Until now the authors of the strong programs YL and MONA have not published the details of their programs' evaluators. If their knowledge becomes available, combining their ideas with MIA IV would probably further increase the playing strength significantly.

Acknowledgements

The authors would like to thank Yngvi Björnsson and Darse Billings for sharing their thoughts about LOA in general, and LOA evaluation functions in particular. We also thank the anonymous referees for their valuable comments.

References

Akl, S. G. and Newborn, M. M. (1977). The principal continuation and the killer heuristic. In *1977 ACM Annual Conference Proceedings*, pages 466–473. ACM, Seattle.

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. Thesis, University of Limburg, Maastricht, The Netherlands.

Billings, D. and Björnsson, Y. (2002). *Mona and YL's Lines of Action Page*. http://www.cs.ualberta.ca/∼darse/LOA.

Björnsson, Y. (2000). Yl wins Lines of Action tournament. *ICGA Journal*, 23(3):179–180.

Björnsson, Y. and Marsland, T. A. (1999). Multi-cut alpha-beta pruning. In Van den Herik, H. J. and Iida, H., editors, *Computers and Games, Lecture Notes in Computing Science 1558*, pages 15–24. Springer Verlag, Heidelberg, Germany.

Björnsson, Y. and Winands, M. (2001). Yl wins Lines of Action tournament. *ICGA Journal*, 24(3):180–181.

Björnsson, Y. and Winands, M. (2002). Yl wins Lines of Action tournament. *ICGA Journal*, 25(3):185–186.

Breuker, D. M., Uiterwijk, J. W. H. M., and van den Herik, H. J. (1996). Replacement schemes and two-level tables. *ICCA Journal*, 19(3):175–180.

Chaunier, C. and Handscomb, K. (2001). Lines of action strategic ideas - part 4. *Abstract Games*, 2(1):12–14.

Donninger, C. (1993). Null move and deep search: Selective-search heuristics for obtuse chess programs. *ICCA Journal*, 16(3):137–143.

Dyer, D. (2000). *Lines of Action Homepage*. http://www.andromeda.com/people/ddyer/loa/loa.html.

Gray, S. B. (1971). Local properties of binary images in two dimensions. *IEEE Transactions on Computers*, 20(5):551–561.

Handscomb, K. (2000). Lines of action strategic ideas - part 1. *Abstract Games*, 1(1):9–11.

Hashimoto, T., Nagashima, J., Sakuta, M., Uiterwijk, J. W. H. M., and Iida, H. (2003). Automatic realization-probability search. Internal report, Dept. of Computer Science, University of Shizuoka, Hamamatsu, Japan.

Heinz, E. A. (1999). Adaptive null-move pruning. *ICCA Journal*, 22(3):123–132.

Knuth, D. E. and Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326.

Marsland, T. A. and Campbell, M. (1982). Parallel search on strongly ordered game trees. *Computing Surveys*, 14(4):533–551.

Sackson, S. (1969). *A Gamut of Games*. Random House, New York, NY, USA.

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

Schaeffer, J. and Plaat, A. (1996). 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.

Uiterwijk, J. W. H. M. and van den Herik, H. J. (2000). The advantage of the initiative. *Information Sciences*, 122(1):43–58.

Winands, M. H. M. (2000). *Analysis and Implementation of Lines of Action*. M.Sc. Thesis, Universiteit Maastricht, Maastricht, The Netherlands.

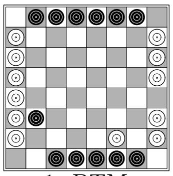
Winands, M. H. M., Kocsis, L., Uiterwijk, J. W. H. M., and van den Herik, H. J. (2002). Temporal difference learning and the Neural MoveMap heuristic in the game of Lines of Action. In Mehdi, Q., Gough, N., and Cavazza, M., editors, *GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation*, pages 99–103. SCS Europe Bvba.

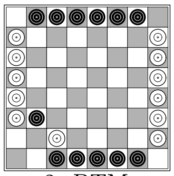
Winands, M. H. M., Uiterwijk, J. W. H. M., and van den Herik, H. J. (2001). The quad heuristic in Lines of Action. *ICGA Journal*, 24(1):3–15.

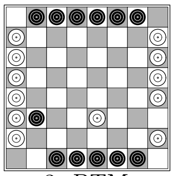
Winands, M. H. M., van den Herik, H. J., Uiterwijk, J. W. H. M., and van der Werf, E. C. D. (2003). Enhanced forward pruning. Accepted for publication.

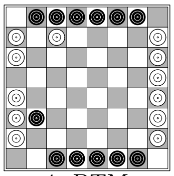
Appendix: Start Positions

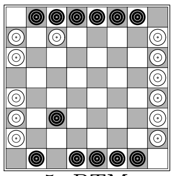
Below the positions are given, which are used in the experiments of Section 4.

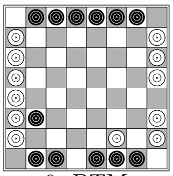

1: BTM

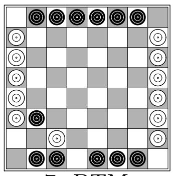

2: BTM

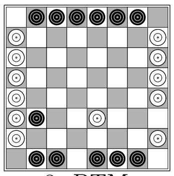

3: BTM

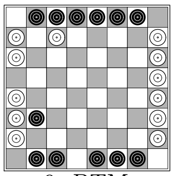

4: BTM

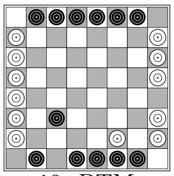

5: BTM


6: BTM


7: BTM


8: BTM


9: BTM


10: BTM