MIA: A World Champion LOA Program

Mark H.M. Winands and H. Jaap van den Herik

MICC-IKAT, Universiteit Maastricht, Maastricht
P.O. Box 616, 6200 MD Maastricht, The Netherlands
{m.winands, herik}@micc.unimaas.nl

Abstract

MIA (Maastricht In Action) is a world-class LOA program, which has won the LOA tournament at the 8th (2003), 9th (2004), and 11th (2006) Computer Olympiad. It is considered as the best LOA-playing entity of the world. In this extended abstract we will present MIA's search engine and evaluation function.

1 Introduction

Around 1975 LOA received its first credits as an AI research topic. For instance, then the first LOA program was written at the Stanford AI laboratory by an unknown author. In the 1980s and 1990s "hobby" programmers wrote several LOA programs. However, all were easily beaten by humans [14]. At the end of the nineties LOA became a clear objective or even target of AI researchers. Considering the role of LOA in the AI domain we may distinguish two different categories.

The first category consists of researchers using LOA as a test domain for their algorithms. Below we mention six telling examples. Eppstein [15] mentioned his dynamic planar-graph techniques to evaluate the connectivity of LOA positions. Kocsis ([22]; [23]) applied successfully his learning time-allocation algorithms and his new move-ordering method in LOA, called the Neural MoveMap heuristic. Moreover, Björnsson [6] confirmed the good results of his multi-cut method for LOA. Up to then the multi-cut was only tested for Chess. Donkers [12] used LOA to test the admissibility in opponent-model search. Sakuta *et al.* [28] investigated the application of the killer-tree heuristic and the λ-search method to the endgame of LOA. These techniques were initially developed for Shogi. Hashimoto *et al.* [19] chose LOA as a test domain for his automatic realisation-probability search method.

The second category consists of researchers trying to build strong LOA programs by using new ideas. For instance, the four programs MIA (Maastricht In Action), BING, YL, and MONA belong to this category. Since 2000 LOA is played at the Computer Olympiad, a multi-games event in which all of the participants are computer programs.

This extended abstract is mainly based on the thesis *Informed Search in Complex Games* [35]. Below we will reveal some of the secrets of the program MIA. It is considered to be the best LOA-playing entity of the world. A demo version of MIA can be played online at the website: http://www.cs.unimaas.nl/m.winands/loa/.

The remainder of the extended abstract is organised as follows. Section 2 explains the game of Lines of Action. The search engine is described in Section 3. In Section 4 the evaluation function is explained. Section 5 provides information on the performance of MIA at the LOA tournaments of the various Computer Olympiads.

2 Lines of Action

Lines of Action (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$\times$8 board. LOA was invented by Claude Soucie around 1960. Sid Sackson [27] described the game in his first edition of *A Gamut of Games*. The objective of a connection game is to group the pieces in such a way that they connect two opposite edges (a static goal) or form a fully-connected group (a dynamic goal). The precise definition of what constitutes a connection depends on the game in question. Whatever the case, the notion of connection became one of the great themes in the world of abstract gaming. Many prominent game inventors made a contribution to this theme. Other examples of connection games are TwixT [9] and Hex [4]. In contrast to the typical connection games, LOA is more chess-like because (1) pieces are moved over the board instead of put on the board, and (2) pieces can be captured.

2.1 The Rules

LOA is played on an 8×8 board by two sides, Black and White. Each side has twelve (checker) pieces at its disposal. Below we are describing the rules used at the Computer Olympiads and at the MSO World Championships. They are formulated in nine rules. In some books, magazines or tournaments, the rules 2, 7, 8, and 9 are different from what is specified here.

1. 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 (see Figure 1a).
2. The players alternately move a piece, starting with Black.
3. 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 1b).
4. A player may jump over its own pieces.
5. A player may not jump over the opponent's pieces, but can capture them by landing on them.
6. 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. Connected pieces are on squares that are adjacent, either orthogonally or diagonally (e.g., see Figure 1c). A single piece is a connected unit.
7. In the case of simultaneous connection, the game is drawn.
8. If a player cannot move, this player has to pass.
9. If a position with the same player to move occurs for the third time, the game is drawn.

The possible moves of the black piece on **d3** in Figure 1b are indicated by arrows. The piece cannot move to **f1** because its path is blocked by an opposing piece. The move to **h7** is not allowed because the square is occupied by a black piece.

2.2 Characteristics

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 is estimated to be $O(10^{64})$ and the state-space complexity $O(10^{23})$ [37]. The game-tree complexity and state-space complexity are comparable with those of Othello [3]. Considering the current state-of-the-art computer techniques, it seems that LOA is not solvable by brute-force methods.

A characteristic property of LOA is that it is a converging game [2], 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 [34], 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.

3 MIA's Search Engine

The standard framework of $\alpha\beta$ search [21] with all kinds of enhancements [24] offers a good start for building a strong LOA-playing program. Thus, MIA started its career with an $\alpha\beta$ depth-first iterative-deepening search. Below we briefly describe MIA's original basic design, which serves as a starting point for our research. Several techniques were implemented to enhance the search. We mention (1) Principal Variation Search (PVS), (2) transposition tables, (3) forward pruning, (4) move ordering, (5) quiescence search, and (6) realisation-probability search.

First, the program uses PVS to narrow the $\alpha\beta$ window as much as possible [25]. This means that the β value equals $\alpha+1$. Such an algorithm is in general more efficient than the original $\alpha\beta$. The basic idea behind the method is that it is cheaper to prove a subtree inferior, than to determine its exact value. It has been shown that this method does well for bushy trees such as occur in Chess. Because the branching factor of LOA (29) is in the same range as that of Chess (38), it works fine in LOA too. Another popular algorithm for searching game trees is NegaScout [26]. The two algorithms (PVS and NegaScout) are essentially equivalent to each other; they expand the same search tree [6].

Second, a *two-deep* transposition table [8] is applied to prune a subtree or to narrow the $\alpha\beta$ window. The well-known Zobrist-hashing method [40] is used for storing the entries in the table. 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 [31].

Third, two forward-pruning techniques are applied. A null move [13] is performed before any other move and it is searched to a lower depth (reduced by *R*) than other moves. At a CUT node a variable scheme, called adaptive null move [20], 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

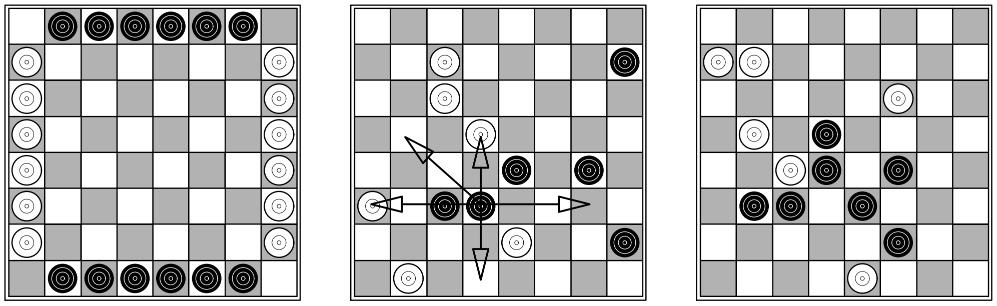

Figure 1: (a) The initial position. (b) An example of possible moves. (c) A terminal position.

2. For ALL nodes $R = 3$ is used. If the null move does not cause a β cut-off, multi-cut [7] is performed. Experiments showed that using multi-cut is not only beneficial at CUT nodes but also at ALL nodes [38].

Fourth, for move ordering, (1) the move stored in the transposition table, if applicable, is always tried first. Then (2) 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: (3) capture moves going to the inner area (the central 4×4 board) and (4) capture moves going to the middle area (the 6×6 rim); finally, (5) all the other moves are ordered decreasingly according to the relative history heuristic [39]. The latter is a new method for move ordering, called the relative history heuristic. It is a combination of the history heuristic [29] and the butterfly heuristic [18]. 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.

Fifth, in the leaf nodes of the tree a quiescence search is performed, since the evaluation function should only be applied to positions that are quiescent. This quiescence search looks at capture moves that form or destroy connections [37] and at capture moves going to the central 4×4 board.

Sixth, the realisation-probability search (RPS) [32] is applied. The RPS algorithm is a new approach to fractional plies. It performs a selective search similar to forward pruning and selective extensions. In RPS the search depth of the move under consideration is determined by the realisation probability of its move category. These realisation probabilities are based on the relative frequencies which are noticed in master games. In MIA, the move categories depend on centre-of-mass, board position, and capturing. In total there are 277 weights.

Finally, we remark that we tested successfully various proof-number search algorithms. PN, PN^2, PDS, and PDS-PN clearly outperformed $\alpha\beta$ in solving endgame positions in LOA. A subject of future research is to find a dynamic strategy which determines when to use proof-number search instead of $\alpha\beta$ in a real-time game.

4 Evaluation Functions

In this section the relevant features of the evaluation function are enumerated and explained. The evaluator consists of the following nine features: *concentration*, *centralisation*, *centre-of-mass position*, *quads*, *mobility*, *walls*, *connectedness*, *uniformity*, and *player to move*. The choice of features that fully cover the description of a position is most relevant. It is better to have all features correct and all the initial weights wrong than to have the initial weights correct and miss one of the (important) features [10]. The description of the features follows below; relevant examples and clarifications are given, adequate references to further details are supplied (Subsections 4.1 to 4.9). It is followed by some information about the use of caching (Subsection 4.10).

4.1 Concentration

The concentration feature is based on the basic principles of threats and solid formations. It measures how close pieces are to each other. By doing so, we reward positions with pieces in each other's neighbourhood. It is hoped that the pieces eventually will be connected in a solid formation or will create threats to win.

The concentration of the pieces is calculated by a centre-of-mass approach (see also the third feature). In MIA it is done in four steps. First, the centre-of-mass of the pieces on the board is computed for each side (in MIA this is done incrementally to save time). Second,

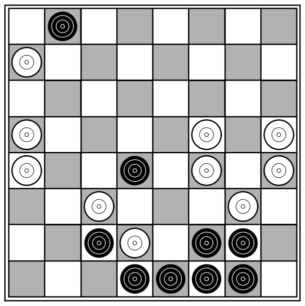

Figure 2: Position with an outlier on **b8**.

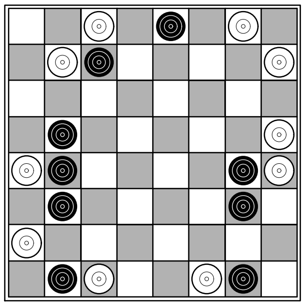

Figure 3: Scattered pieces.

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. The sum-of-minimal-distances is dependent on the number of pieces on the board (see the example below) and 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.

The disadvantage of this feature 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 2 the black pieces are grouped around their centre-of-mass at **e2**, but the black piece on **b8** is rather far away from this group.

4.2 Centralisation

According to the centralisation feature pieces controlling the centre are more important than others. Centralisation is important because pieces have to move through the centre to connect with each other.

Analogous to piece-square tables in Chess, each piece obtains a value dependent on its board square in MIA. 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 (but see Subsection 4.6). 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.

We remark that centralisation can also be obtained indirectly by punishing moves not going to the centre. This is done in the mobility feature.

4.3 Centre-of-Mass Position

The centre-of-mass position feature is indirectly based on the basic principle of solid formations. It evaluates the global position of all the pieces. This means that it looks at the position of the centre-of-mass on the board. The initial idea was to prevent formations from being built on the edges, where they are rather easily destroyed or blocked.

The value of this feature is dependent on the board square of the centre-of-mass. We use a simple table lookup for computation in MIA. Interestingly, after applying Temporal-Difference (TD) learning to enhance the weights, the weight for the centralised centre-of-mass feature changed its sign [36], 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 3). 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. Therefore, this feature contributes to the concentration and indirectly to the connectedness (see Subsection 4.7). 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 at that place, whereas groups residing closer to the edge can only be attacked from one side.

4.4 Quads

The quads feature is based on the basic principles of solid formations and material advantage. It looks at the solid-

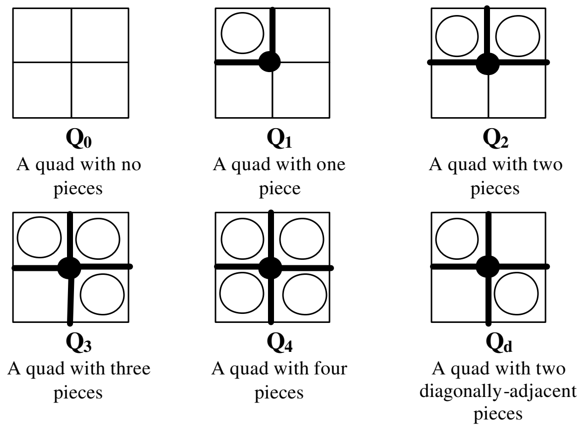

Figure 4: Six different quad types.

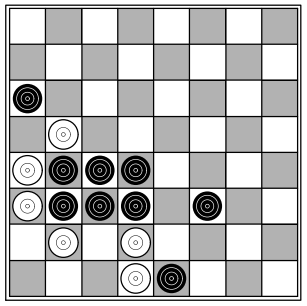

Figure 5: Position with two black Q_4 's.

ness of the formation in particular. The feature favours pieces, which are connected in more than one direction, because it is harder for the opponent to disconnect them. 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.* [37]. The heuristic is based on the use of quads, an Optical Character Recognition method. A quad is defined as a 2×2 array of squares [16]. In LOA there are 81 quads for each side, including also quads covering only a part of the board along the edges. Taking into account rotational equivalence, there are six different quad types, depicted in Figure 4.

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 reward only Q_3's and Q_4's, which are at a distance of at most two squares of the centre-of-mass. For instance, Black has two Q_4's in Figure 5. In passing we note that this feature implicitly favours a material advantage.

The effect of implicitly favouring a component due to the introduction of another feature is first described by Schaeffer [30] for chess. Obviously, it is a challenge to analyse the interrelationship in LOA too, since it turns out to be an issue for almost all components. A possible disadvantage of this feature is that if the position becomes too solid, its flexibility may decrease drastically. The mobility feature may adjust this disadvantage.

4.5 Mobility

The mobility feature looks at the potential of the moves in a position. The idea is that it is easier to connect your own pieces or obstruct the connection of opponent pieces if you have more and better moves. The feature was first implemented in MONA and YL.

When evaluating a position in MIA, the possible moves of both sides are generated (irrespective of who is to move). The moves are not rewarded equally. Experiments have shown that certain move types are to be preferred above others (see also [19]). 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 feature are not high. For each line configuration (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.

An advantage of this feature that it is fast to evaluate. A disadvantage of this implementation is that it is too static. For example, all capture moves are given a bonus, even the ones which capture the last unconnected opponent's piece. Moreover, all edge moves are given a penalty, even if they connect to the main group. A more global look of the position would be needed to distinguish these kind of exceptions.

4.6 Walls

The wall feature is based on the basic principle of blocking. 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

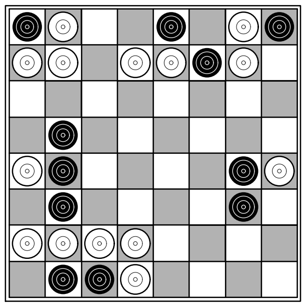

Figure 6: Position with walls.

win. Even partial blocking, called a wall [17], 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 stored in a table. Using a bit-board representation they can be easily looked-up. We remark that we take special care of walls which block corner pieces.

For example, in Figure 6 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. 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. The pieces on **b1** and **c1** are completely blocked, but we take only the two 3-centre-directions blocks into account. Thus, we only look at certain blocking patterns for edge pieces.

4.7 Connectedness

The connectedness feature is based on the basic principles of threats and solid formations. It measures the pairwise connections between the pieces. We reward positions with high connectedness; it is hoped that they eventually will be connected in one unit or will create threats to win.

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 feature in LOA evaluation functions can be dangerous because the program might wildly capture pieces. This feature does not take into account whether a connection is important. To distinguish among connections, 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 feature.

Of course the connectedness feature is highly correlated with the concentration feature and the quads feature. Though each has its own merits, these three features should be carefully tuned consequently.

4.8 Uniformity

The uniformity feature is based on the basic principle of solid formations. It is used to achieve a uniform distribution of the pieces [11] to counterbalance the negative effects of the centre-of-mass approach. It prevents that one or more pieces become too remote from the main group.

In MIA this is done in a way which is primitive but effective. The smallest rectangular area covering the distributed pieces is computed. The smaller the area is, the higher the reward is. An analogous implementation was first realised in the program YL [5].

4.9 Player to Move

The player-to-move feature is based on the basic principle of the initiative. It rewards the moving side. Having the initiative is mostly an advantage in LOA [34] like in many other games [33].

Since MIA is using variable-depth search(because of the adaptive null move, the multi-cut, and quiescence search) not all leaf nodes are evaluated at the same depth. Therefore, leaf nodes in the search tree may have a different player to move, which is compensated in the evaluation function. This is done by giving a small bonus to the side to move.

4.10 Caching certain Features

It is possible in our evaluation function to cache computations of certain features, which can be used in other positions. For example, let us assume that we investigate the move **b8-c8** in Figure 2 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 is easy to see that we do not have to compute the concentration, centralisation, centre-of-mass position, quads, connectedness, and uniformity for White again. Evaluation of these six features, which are independent of the position of the other side, are stored in an evaluation cache table. In the current evaluation function this gives a speed-up of at least 60 per cent in the number of nodes investigated per second.

5 MIA at the Computer Olympiads

MIA has participated in six Computer Olympiads. In 2000, MIA I finished third at the 5th Computer Olympiad. In retrospect its evaluator was primitive, although MIA I won a game against the runner-up MONA and the winner YL. In 2001, the tournament program (MIA II) shared the first place with YL in the regular tournament at the CMG 6th Computer Olympiad. The play-off match was won by YL. The improved MIA III finished second at the 7th Computer Olympiad in 2002. The program scored 1.5 points out of 4 against the much improved winner YL. MIA IV won the 8th Computer Olympiad in 2003, losing only one game against BING. At the 9th Computer Olympiad (2004) MIA 4.5 won the the tournament scoring 11.5 points out of 12. In Turin, Italy, 2006, MIA 4.5 won the 11th Computer Olympiad with a perfect score.

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 Press, New York, NY, USA, 1977.

[2] L.V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Rijksuniversiteit Limburg, Maastricht, The Netherlands, 1994.

[3] L.V. Allis, H.J. van den Herik, and I.S. Herschberg. Which games will survive? In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2: the Second Computer Olympiad*, pages 232–243. Ellis Horwood, Chichester, England, 1991.

[4] V.V. Anshelevich. A hierarchical approach to computer hex. *Artificial Intelligence*, 134(1-2):101–120, 2002.

[5] D. Billings and Y. Björnsson. Search and knowledge in lines of action. In H.J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games 10: Many Games, Many Challenges*, pages 231–248. Kluwer Academic Publishers, Boston, MA, USA, 2003.

[6] Y. Björnsson. *Selective Depth-First Game-Tree Search*. PhD thesis, University of Alberta, Edmonton, Canada, 2002.

[7] 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.

[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. Bush. An introduction to TwixT. *Abstract Games*, 1(2):9–12, 2000.

[10] S. Bushinsky, 2004. Personal Communication.

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

[12] H.H.L.M. Donkers, J.W.H.M. Uiterwijk, and H.J. van den Herik. Admissibility in opponent-model search. *Information Sciences*, 154(3-4):119–140, 2003.

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

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

[15] D. Eppstein. Dynamic Connectivity in Digital Images. *Information Processing Letters*, 62(3):121–126, May 1997.

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

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

[18] D. Hartmann. Butterfly boards. *ICCA Journal*, 11(2-3):64–71, 1988.

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

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

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

[22] L. Kocsis, J.W.H.M. Uiterwijk, and H.J. van den Herik. Learning time allocation using neural networks. In T.A. Marsland and I. Frank, editors, *Computers and Games*, *Lecture Notes in Computer Science 2063*, pages 170–185, Berlin, Germany, 2001. Springer-Verlag.

[23] 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, Vol. 2070*, pages 45–50. Springer-Verlag, Berlin, Germany, 2001.

[24] T.A. Marsland. A review of game-tree pruning. *ICCA Journal*, 9(1):3–19, 1986.

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

[26] A. Reinefeld. An improvement to the Scout search tree algorithm. *ICCA Journal*, 6(4):4–14, 1983.

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

[28] M. Sakuta, T. Hashimoto, J. Nagashima, J.W.H.M. Uiterwijk, and H. Iida. Application of the killer-tree heuristic and the lamba-search method to lines of action. *Information Sciences*, 154(3–4):141–155, 2003.

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

[30] J. Schaeffer. The relative importance of knowledge. *ICCA Journal*, 7(3):138–145, 1984.

[31] 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.

[32] Y. Tsuruoka, D. Yokoyama, and T. Chikayama. Game-tree search algorithm based on realization probability. *ICGA Journal*, 25(3):132–144, 2002.

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

[34] M.H.M. Winands. Analysis and implementation of Lines of Action. Master's thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2000.

[35] M.H.M. Winands. *Informed Search in Complex Games*. PhD thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2004.

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

[37] 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.

[38] 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*, 175(4):315–329, 2004.

[39] M.H.M. Winands, E.C.D. van der Werf, H.J. van den Herik, and J.W.H.M. Uiterwijk. The relative history heuristic. In H.J. van den Herik, Y. Björnsson, and N.S. Netanyahu, editors, *Computers and Games*, *Lecture Notes in Computer Science 3846*, pages 262–272, 2006.

[40] 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*, Vol. 13, No. 2, pp. 69–73.