Analysis and Implementation of Lines of Action

Mark Winands and Jos Uiterwijk

Department of Computer Science, Universiteit Maastricht, The Netherlands

Abstract

LOA is a two-person zero-sum game with perfect information. It is a connection game, albeit non-typical. The state-space complexity and game-tree complexity are comparable with those of Othello. For the state of the art of computer techniques, LOA seems uncrackable. An alpha-beta depth-first iterative-deepening searching program (MIA) is developed. Moves are generated for a part incrementally. A heuristic is used that can be used to show when positions in LOA are not terminal. This heuristic is based on quad counts and euler numbers. Several move-ordering techniques are used: transposition-table moves, killer moves, and history heuristic work very well in LOA. Some explored game-specific ordering techniques are less successful. PVS and null moves works fine. Although LOA is a connection game, capturing pieces is important. Therefore a quiescence search is performed, which considers only capture moves destroying or creating connections.

Concentration, centralisation, solid formations, connections and (partial) blocking are important components in evaluation functions for LOA. Based on these concepts, three different evaluation functions have been constructed, denoted *normal*, *quad* and *blocking*. They all consider the concentration and the centralisation of the pieces on the board. The quad evaluator in addition uses quad counts to determine the solidness and connectioness of the formations, whereas the blocking evaluator takes into account the mobility of the pieces towards the neighbourhood of the centre of mass. Experiments show that the quad evaluator performs best.

1. Introduction

Lines of Action (LOA) is a board game which is played on a checkers board. It is a two-person zero-sum game with perfect information. LOA is a connection game, albeit non-typical. Claude Soucie invented it around 1960. Sid Sackson (1969) described it in his first edition of *A Gamut of Games*. We use the rules described in the *second edition* of A Gamut of Games:

1. One player controls the twelve black pieces and the other the twelve white pieces. The black pieces are placed in two rows along the top and bottom of the board, while the white stones are placed in two files at the left and right of the board. The set-up of the game is shown in figure 1.
2. Black moves first.
3. Each turn, the player to move has to move one of his pieces, in a straight line, exactly as many squares as there a pieces of either colour *anywhere* along the line of movement. (These are the *Lines of Action*).
4. You may jump over your own pieces.
5. You may not jump over your opponent's pieces, but you can capture them by landing on them.
6. The object of the game is to turn your pieces into one connected unit. The first player to do so is the winner. The connections within the group may be either orthogonal or diagonal. For example, in figure 2 black has won because his pieces form one connected unit.
7. If one player is reduced by captures to a single piece, that is a win for this player.
8. If a move simultaneously creates a single connected unit for both the player moving and the opponent, the player moving wins.

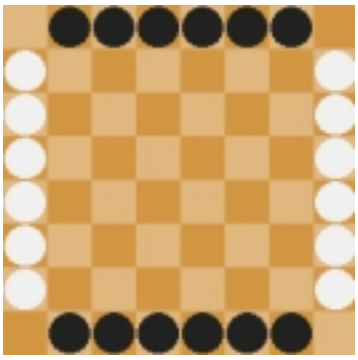

Figure 1: Board set-up

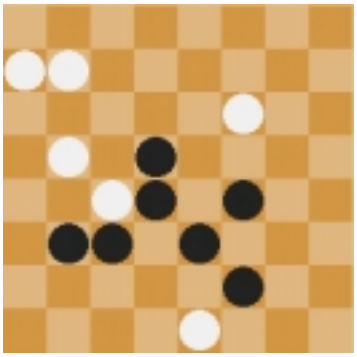

Figure 2: A terminal position.

These are the official rules. Some situations were not covered by the original rules. For these situations, the rules of the Mind Sports Olympiad (MSO) are used:

9. If a player cannot move, this player loses.[1]
10. If a position with the same player to move occurs for the second time, this is a draw.

The notation concerning board and moves used in this article is the standard chess notation. The possible moves of the black piece on **d3** in figure 3 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 a black piece is already there.

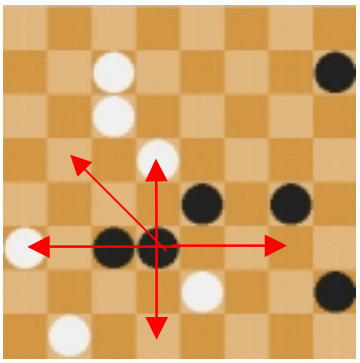

Figure 3: Movement of pieces.

2. Complexity of LOA

An upper bound of the state-space complexity for LOA can easily be derived. The computation is of the same kind as in checkers (Schaeffer and Lake, 1996). Let *B* be the number of black pieces and *W* the number of white pieces. There are at most twelve black and twelve white pieces. Each of the 64 squares on the board can be occupied by a piece. The number of positions having 24 pieces or less is derived in the following formula:

[1] The use of this rule is disputable, because some LOA players think that the player has to pass.

$$\sum_{B=1}^{12}\sum_{W=1}^{12} Num(B,W) - Num(1,1) \qquad \text{where} \quad Num(B,W) = \binom{64}{B}\binom{64-B}{W}$$

In the formula we have neglected all positions with two pieces or fewer, because they are impossible to reach. One of the players would have won before the last move was made. For the same reason we neglect all positions with only pieces of the same colour on the board. These neglections only lead to a reduction of 8.4×10^{12} positions.

In table 1, we have calculated the separate combinations per number of pieces. For example, the combination of three pieces is *Num(2,1) + Num(1,2).* If we sum the numbers for 3 up to and including 24 pieces together, we obtain 1.3×10^{24} different positions. Of course we have still included some positions, which are not reachable from the start (Dyer, 2000), but we are convinced that this lowers the state-space complexity only marginally. Therefore, we take for the state-space complexity $O(10^{24})$.

# of pieces	**# of positions**
3	249984
4	8895264
5	228735360
6	4648410816
7	78273240192
8	1124246003472
9	14045698101120
10	154805625541952
11	1521396969611904
12	13445574994311264
13	107590873672114560
14	782632117126476864
15	5188514989534830720
16	31335094798158420360
17	171930216128039066880
18	853283294857675368960
19	3807935897946939333120
20	15158514055288777729920
21	53164643498259191458560
22	160592373542262268414080
23	396757628751471486670080
24	677794282450430456394720
Total:	1308338020676454019380152

Table 1: State-space complexity.

The game-tree complexity can be approximated by the size of a uniform game tree with as depth the average game length and as branching factor the average number of legal moves (Allis, 1994). Lacking a large database with LOA games we have approximated these numbers using experiments in which our program plays against itself. The results gave an average game length of 38 ply and an average branching factor of 30. The game-tree complexity thus is about $O(10^{56})$. The state-space complexity and game-tree complexity are comparable with those of Othello. For the state of the art of computer techniques, LOA is uncrackable. Another curiosity of LOA is that its terminal positions still have a lot of pieces remaining on the board. Therefore endgame databases are probably not very useful in LOA.

3. The Quad Heuristic

In LOA it is not trivial to detect a terminal node. Remember that the game is over when a group of pieces with the same colour is connected. In most cases this is not easy to detect. Checking on a game-over situation is a time-consuming process in LOA, contrary to games like draughts and chess. The obvious solution is to incrementally update the count of pieces when moves are made. Next, count the number of pieces of an arbitrary group, using a depth-first algorithm. If this number equals the total number of stones of that colour, the game is over. However, the depth-first search necessary to retrieve the number of pieces is a time-consuming process.

Fortunately, there is a heuristic that can detect in 99% of the cases that a game definitely is not over. The solution lies in using bit quads, an OCR method. A quad is defined as a 2×2 array of bit cells. In LOA there are 81 possible bit quads for each side, including also bit quads covering only a part of the board along the edges. Considering rotational equivalence, there are six distinct types, depicted in figure 4:

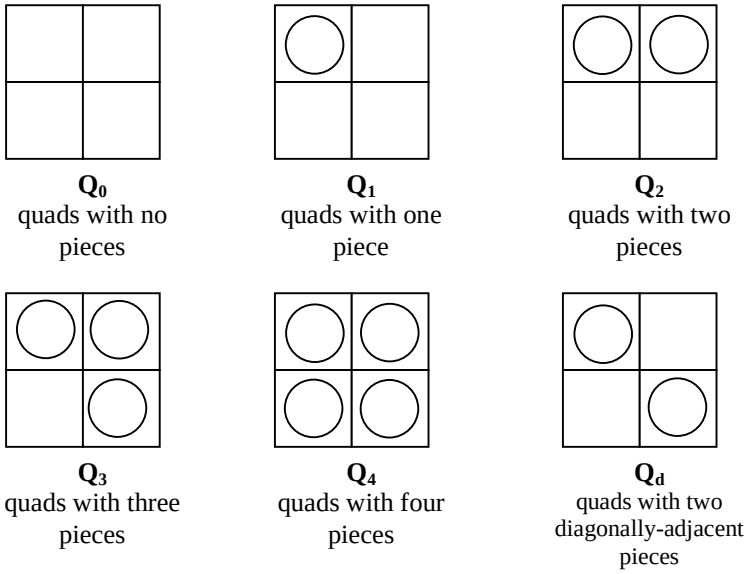

Figure 4: Different kinds of quads.

Notice, that we have two kinds of quads with two pieces. One in which the pieces are diagonally adjacent, another in which they are orthogonally adjacent. If a quad only partially covers the board, the cells not covering the board are considered empty. Summing over all quads gives us the euler number *E* of the board as follows:

$$E = \left(\sum Q_1 - \sum Q_3 - 2\sum Q_d\right)/4$$

For a motivation of this formula see (Gray, 1971). The euler number represents the number of connected groups minus the number of holes (interior regions). For example, in figure 5 the euler number of black is three, because black has three connected units. The euler number of white is one. White has two connected units, but his pieces are surrounding the region at **e5**. This is called a hole. Therefore the euler number of white is equal to two connected units minus one hole. This example

shows us that euler numbers do not completely detect win positions. If the euler number is greater than one, the game is definitely not over (Minsky and Papert, 1988). Only when there are at least as many holes as connected groups, we have to use another method. But fortunately holes in LOA are rare and using this heuristic is often enough to detect non-terminal positions.

Figure 5: Position with holes.

An advantage of this method is that we incrementally can update the bit quads and their counts. There are mostly eight quads that change as the result of a move, with a maximum of twelve when a capture move occurs. Although the bookkeeping is not trivial, it is still cheap enough to outperform other methods.

In table 2 we see that using bit quads causes a speed up of 10 a 15 % at the start position of the game. An argument against the bit quad method is that its performance is low when the total number of stones is also low, but LOA usually ends with many stones still on the board. The big gain is that we can use quad counts in the quiescence search and in the evaluation function also.

Depth	Time (ms) with quads	Time (ms) without quads	Nodes
1	50	60	37
2	220	220	217
3	440	440	1671
4	990	1100	6081
5	6530	8070	105933
6	52120	60470	817567

Table 2: Results of using quads.

4. Evaluation Functions

Although knowledge about LOA is not well developed, there are some basic principles when playing LOA. Carl von Blixen (2000) described on his homepages some guidelines, which we will take as a framework for developing an evaluation function.

4.1 General principles of LOA

- *Create a threat as quickly as possible.*

If a player can possibly connect all his pieces in one connected unit, this situation is called a *threat*. Also as the opposite party can prevent the connection to be made, the situation is still a threat, because the player *threats* to win. The first player to create a threat to win has a big advantage, because the opponent will have to break up his own formation to stop this threat. In figure 6 we see that white can win by the move **e8-e6**. Black can only prevent this by **g4×e4**, but he has to weaken his own formation.

Figure 6: Threat position.

Figure 7: Blocked piece.

- *Block opponent's pieces*

Because one is not allowed to jump over opponent's pieces, it is possible to *block* a piece. A piece is *completely* blocked, if it cannot move anywhere. Blocking a piece far away from the rest can be very effective. In figure 7 the opposite side has to waste many moves to first capture pieces around the blocked one at **a8** and try to connect again. During that time, black is likely able to create a threat. Even partial blocking can be very effective if it forces the opponent to find a way around your pieces. Handscomb (2000a) describes that it is a common tactic to create a wall at the b- or g-file, or the 2nd or 7th row in the beginning phase of the game, which partially blocks pieces of the opposite side.

- *Connect pieces on multiple ways*

If pieces are connected in more than one direction it is harder for the opponent to break them up. In figure 8 black has created a rock solid formation in the centre of the board, which is hard to break up. The disadvantage is that it is hard to create this kind of formation fast. The danger exists that the opponent can make a threat first. Mostly it is good practise first to look if it is possible to connect a piece to a group of connected pieces and only secondly to see if a solid connection can be made.

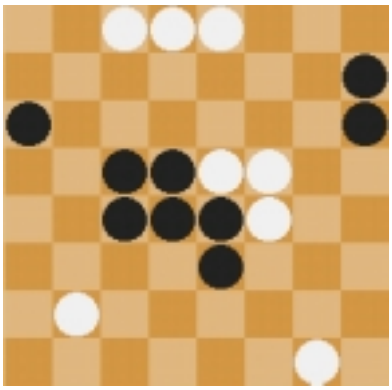

Figure 8: Solid formation.

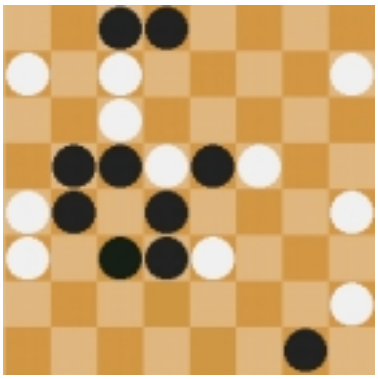

Figure 9: Roessner vs. Handscomb.

These principles are good to have in mind when playing LOA. But they only say *what* is good in LOA, not how to *achieve* it. Some programmers have implemented a centralisation factor in their evaluator. Pieces in the centre are more awarded than pieces outside. Handscomb (2000b) argues that the benefits of centralisation in LOA are exaggerated. Because of the nature of the game, pieces huddled together in the middle are incapable of capturing each other. Even with only two pieces in a given line of action you need a distance of two squares for a capture, and if there are three pieces a considerable distance from the edge to the middle of the board is required. In figure 9 some pieces of white are scattered around the edges of the board. The white piece at **h4** can destroy blacks formation in the middle by capturing the black piece **d4**. The white pieces at the edges of the board are acting as cruise missiles. This is not possible when all the pieces are in the centre.

The last aspect we want to discuss in this subsection is capture moves. At first sight capturing opponent's pieces is not a good idea. However, it is not hard to see that capture moves in the endgame often are not bad. First, sometimes the only defence against a hostile threat is capturing a piece such that its formation is destroyed. Second, it also happens that the only way one is able to connect with its own formation is by a capture move. Third, a capture move is the only way to free one's blocked piece. Some authors argue that capturing pieces in the begin and middle game is also good. They think that a material advantage like in chess exists, because if a player has more pieces than his opponent, he has more chances to prevent hostile threats and create his own threats. The opponent has fewer possibilities to prevent threats. Whether material is really an advantage depends on the position of the pieces on the board. A material component in the evaluation function is dangerous in LOA, because the program might wildly capture pieces. A more subtle method has to be used, because sometimes a material disadvantage can be an advantage.

4.2 LOA evaluators

It is important that the evaluator is fast and a good estimator. The problem is to find those characteristics of LOA, which make the program play well. We have implemented three evaluators: *normal*, *quad* and *blocking*.

- *Normal evaluator*

This is the core of every evaluator implemented in the program. We have chosen for a centre-of-mass approach. For each side, the centre of mass of the pieces at the board is computed. Next we compute for each piece its distance to the centre of mass. The distance is measured as the minimal number of squares the piece has to travel going to the centre of mass. Subsequently the average distance towards the centre of mass is calculated. Because only one piece can be in the centre of mass at the time, boards with few pieces are favoured. Therefore a recalculation is done which takes the number of pieces into account. The inverse of this average distance is defined as the concentration.

How lower the average distance of the centre of mass, how higher the concentration. We favour boards where the pieces of the same colour are more in proximity of the centre of mass. Notice that we do not look at connections. Because we favour boards where pieces are in the neighbourhood of each other, eventually they will be connected *in multiple ways*. This evaluator favours sometimes capture moves. For example, if **1. d1-d3** is played**,** the program will respond with **a3×d3**. This move will not only improve its own concentration, but will also undermine the opponent's concentration, its pieces being even more scattered around the board.

Also penalties are given for each piece at the edge, because they are easy to block. Finally positions with a more centralised centre of mass are slightly favoured, preventing formations built at the edges of the board, which are easy to break up or to block.

- *Quad evaluator*

This evaluator is called quad, because it uses the quad count. We can use the quad count on several ways. For example, we can easily compute the euler number of each side. If we assume that holes do not occur, we have also computed the number of connected groups. It seems wise to take this factor into consideration. Because the goal of the game is to create one connected group, minimising the number of connected groups is a good subgoal of the game. However, at the start of the game this number is only two. Only stupid moves can keep this number at two. Every sequence of normal moves in the beginning phase of the game will increase this number.

The *normal* evaluator has no notice of solid formations. Formations with quads of three or four pieces are not possible to break up by a single capture. It seems reasonable to favour boards with many Q_3's and Q_4's. The danger exists that a lot of those quads are created outside the neighbourhood of the centre of mass. Therefore only Q_3's and Q_4's are rewarded, which lie at a distance of at most two of the centre of mass. Notice, that this evaluator implicitly favours a material advantage.

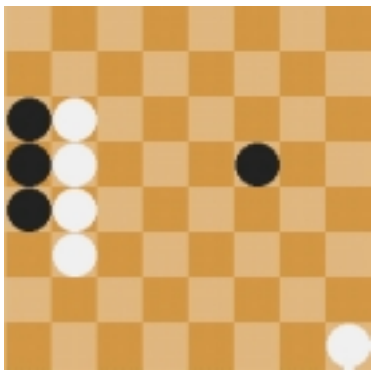

Figure 10: Wall position.

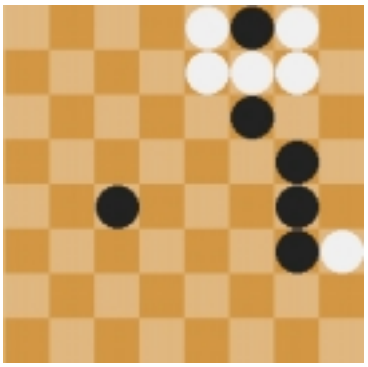

Figure 11: Pseudo blocking.

- *Blocking evaluator*

The problem with the previous two evaluators is that they are not able to detect blocking situations. For example, in figure 10 black is to move. If he plays **f5-c5,** this will result in a better value of the evaluation function then doing **f5-e6**, but the piece at **c5** can not move any further to the centre of mass because of the white wall. However, the piece at **e6** is still able to move further. Therefore we have to take into account if pieces are partially blocked. If a piece outside the neighbourhood of the centre of mass cannot move, a penalty is given. Also lower penalties are given for pieces, which have two moves or less. For pieces outside the region of the centre of mass is looked if they are able to move directly to

the region of the centre of mass. For this evaluator we need to use our move generator. This is the reason why this evaluator is much slower than the others. If the benefit of better evaluations will outperform the disadvantage of searching less deep is doubtful. This evaluator has a notion of blocking but has his own faults: sometimes a piece can easily be freed by a capture move and be connected. For example, in figure 11 the black piece at **f8** is completely blocked. This board position is too negatively evaluated, because black can free this piece and win the game by **c4×f7**. Luckily, this kind of situation can easily be detected by the quiescence search.

5. Some Results

We have performed some experiments for the three evaluators described in the previous section. The experiments were performed on a P366 with 128 Kb SDRAM. All evaluators played 200 games against each other, switching sides halfway. The program uses quiescence search, a 2^{14}-entry transposition table, the history heuristic and killer moves, but no null moves. The time was limited to 5 sec. per move. The match results are given in table 3.

Quad vs. Normal	Quad vs. Blocking	Normal vs. Blocking
127-71-2	119-76-5	117-81-2

Table 3: Comparing the evaluators with each other (wins-losses-draws).

We can conclude from the table that the quad evaluator is the best of the three. Interestingly, the quad evaluator defeats the blocking evaluator with no significantly higher numbers than the normal evaluator does. Because the normal evaluator defeats the blocking evaluator and is defeated by the quad evaluator, one should expect a glorious victory for the quad evaluator when playing against blocking. Possibly, both evaluators take advantage of the same weaknesses of the blocking evaluator.

When the normal and quad evaluators are used, the speed of the search performed is between 17000 and 18000 nodes per second (nps). If the blocking evaluator is used a speed of only 5000 nps is achieved. Thus, the blocking evaluator is a much slower evaluator, causing that normally roughly a ply less can be searched.

Although the goal of the study is not to make the best LOA program in the world, we compared our program against others. We have only used recent programs playable at an IBM compatible PC. A description of the programs we used is given below.

- **Loa2D.** Benjamin Guihaire has written this program. The version currently used is 3.1, released in May 2000. The program is considered as lightweight. It has different evaluators, our program having played against the *Normal* evaluator.
- **LoaW.** Dave Dyer and Ray Tayek have created this program. It was released in 1996. The program is written in C++.
- **Mona.** Darse Billings, who is a member of The University of Alberta Games Group, has created this program. The program was released in 2000. Mona has won of the best LOA players in the world. Still, the author of Mona feels it is too early to declare over-champion (significantly stronger than the human world champion) status for Mona. The program is written in C.
- **YL.** Yngvi Bjornsson, who is also a member of The University of Alberta Games Group, has constructed this program. This program is normally better than Mona, but in a 30-minutes-per-move match Mona had the upperhand. It searches deeper than Mona does. YL is written in C.

Our program MIA (Maastricht In Action) played with both colours against these programs. We used the same settings as for the evaluator experiments, but this time with a thinking time of 30 sec. and using the null-move heuristic. The results are given in the table 4.

MIA vs. Loa2D[2]	MIA vs. LoaW[3]
2-0	2-0
MIA vs. Mona[4]	MIA vs. YL[4]
2-0	0-2

Table 4: Comparing MIA with other programs.

6. Conclusions and Future Research

LOA is a game with complexities comparable to those of Othello. It seems that it will be "safe" for being cracked for the near future. Therefore, we have put emphasis on the development of a strong game-playing program. So far, we have built in a relatively short time a program, MIA, which is reasonably strong. Especially the use of the quad heuristic, both in detecting terminal positions and as an important component in the evaluation function is responsible for MIA's strength.

We expect that the strength of MIA can considerably be enhanced in the near future. One way of course is to extend and fine-tune the alpha-beta enhancements. Moreover, we will investigate the use of endgame databases. However, most games played end with a considerably number of stones on the board, which means that the effect of endgame databases will probably not be very large. Finally, the best prospects to enhance MIA's playing strength will be the addition of domain knowledge in the evaluation function. The playing and understanding of games is therefore a necessary prerequisite, for which the organisation of events like the Computer Olympiad is essential.

References

1. Allis, L.V. (1994). Searching for Solutions in Games and Artificial Intelligence. Ph.D. thesis Rijksuniversiteit Limburg, Maastricht, The Netherlands.
2. Blixen, C. von (2000). Lines-of-Action. http://www.student.nada.kth.se/~f89-cvb/loa.html.
3. Dyer, D (2000). Lines of Action Homepage. http://www.andromeda.com/people/ddyer/loa/loa.html.
4. Gray, S.B. (1971). Local Properties of Binary Images in Two Dimensions. *IEEE Transactions on Computers*, Vol. C-20, No. 5, pp. 551-561.
5. Handscomb, K. (2000a) Lines of Action Strategic Ideas – Part 1. *Abstract Games.* Vol. 1, No. 1.
6. Handscomb, K. (2000b) Lines of Action Strategic Ideas – Part 2. *Abstract Games.* Vol. 1, No. 2.
7. Minsky, M. and Papert, S. (1988). Perceptrons: An Introduction to Computational Geometry. MIT press, Cambridge, MA, USA.
8. Sackson, S. (1969). *A Gamut of Games*. Random House, New York, NY, USA.
9. Schaeffer, J. and Lake, R. (1996). Solving the Game of Checkers. *Games of No Chance* (ed. J. Nowakowski). *MSRI Publications*, Vol. 29, pp. 119-133. Cambridge University Press, Cambridge, UK.

[2] In Loa2D it is not possible to set a certain thinking time, only a certain level. The level was set at 6 (hard), because it takes Loa2D approximately 30 sec. thinking time.
[3] In LoaW it is not possible to set a certain thinking time, only a certain level. The level of Loa2D was therefore set at 5 and the thinking time of MIA at 5 sec.
[4] Mona and YL do only run at the server of University of Alberta.