24. The program makes the correct move after searching 63 plies then moving instantly, reporting a score of draw for the strong side. This is the same phenomenon that was observed in the other position, and the repetition spiral is about to begin.
25. 21-25 wins in 181, but 17-14 allows White 4 additional plies. Cumulative slip = 36 plies.
26. 21-25 wins in 183, but 14-09 allows White 8 additional plies. Cumulative slip = 44 plies.
27. Another instance of hitting the limit of 63 plies of search due to hash table saturation. This is the correct move to win optimally but the program is reporting a draw from the repetitions.
28. Although this is the correct move for the optimal win, the program searched for over four times as long to reach ply 31 on this move than for the average of the previous moves.
29. 13-17 wins in 181, but 13-09 allows White 4 additional plies. Cumulative slip = 48 plies.
30. 17-22 wins in 175, but 19-24 allows White 4 additional plies. Cumulative slip = 52 plies.
31. 17-21 wins in 177, but 17-14 allows White 4 additional plies. Cumulative slip = 56 plies.
32. 14-17 wins in 179, but 19-23 allows White 4 additional plies. Cumulative slip = 60 plies.
33. The classic problem of "wandering Kings" is present here. Now 45 moves into the game, with 31 moves of regression, the program has only advanced 14 full moves towards the goal. This was, by far, the longest time required to complete a 31 ply, search at 27 minutes 16 seconds.
34. Making the correct move, and spending only 47 seconds to complete 31 plies of search in this instance.
35. 18-22 wins in 171, but 24-27 allows White 4 additional plies. Cumulative slip = 64 plies.
36. 22-25 wins in 173, but 22-17 allows White 4 additional plies. Cumulative slip = 68 plies.
37. 18-22 wins in 175, but 18-14 allows White 4 additional plies. Cumulative slip = 72 plies.
38. 14-18 wins in 177, but 19-23 allows White 4 additional plies. Cumulative slip = 76 plies.
39. 10-14 wins in 177, but 23-19 allows White 4 additional plies. Cumulative slip = 80 plies.
40. Another long search, indicative of opportunities to give up the draw appearing in the anticipated line of play. 10-14 wins in 179, but 08-11 allows white 8 additional plies. Cumulative slip = 88 plies.
41. 10-06 wins in 185, but 22-17 allows White 4 additional plies. Cumulative slip = 92 plies.
42. Making the correct move after a research on ply 31, replacing the "wandering" 06-02 move.
43. 14-17 wins in 175, but 14-10 allows White 4 additional plies. Cumulative slip = 96 plies.
44. 10-14 wins in 177, but 12-08 allows White 4 additional plies. Cumulative slip = 100 plies.
45. 10-14 wins in 179, but 08-11 allows White 8 additional plies. Cumulative slip = 108 plies.
46. 10-06 wins in 185, but 22-17 allows White 4 additional plies. Cumulative slip = 112 plies.
47. 10-06 wins in 185, but 10-07 allows White 4 additional plies. Cumulative slip = 116 plies.
48. 14-17 wins in 175, but 23-27 allows White 4 additional plies. Cumulative slip = 120 plies.
49. 22-25 wins in 173, but 12-08 allows White 8 additional plies. Cumulative slip = 128 plies.
50. 08-12 wins in 177, but 08-11 allows White 4 additional plies. Cumulative slip = 132 plies.
51. 14-17 wins in 179, but 22-17 allows White 4 additional plies. Cumulative slip = 136 plies.
52. 08-12 wins in 175, but 17-14 allows White 4 additional plies. Cumulative slip = 140 plies.
53. 14-17 wins in 175, but 14-10 allows White 4 additional plies. Cumulative slip = 144 plies.
54. 10-14 wins in 177, but 12-08 allows White 4 additional plies. Cumulative slip = 148 plies.
55. 10-14 wins in 179, but 08-11 allows White 8 additional plies. Cumulative slip = 156 plies.
56. 10-06 wins in 185, but 22-17 allows White 4 additional plies. Cumulative slip = 160 plies.
57. As was seen in the position at [42], here too the correct move was made after a research on ply 31, replacing the "wandering" 06-02 move.
58. Another correct move, played here at move 94, leads to the same position at move 60, which was 68 plies ago.
59. 14-17 wins in 175, but 14-10 allows White 4 additional plies. Cumulative slip = 164 plies. The position here at move 97 is the same as was seen at move 63, playing in the cycle from 68 plies ago. See note [43].
60. 10-14 wins in 177, but 12-08 allows White 4 additional plies. Cumulative slip = 168 plies. The position here at move 98 is the same as was seen at move 64, playing in the cycle from 68 plies ago. See note [44].
61. 10-14 wins in 179, but 08-11 allows White 8 additional plies. Cumulative slip = 176 plies. The position here at move 99 is the same as was seen at move 65, playing in the cycle from 68 plies ago. See note [45].
62. 10-06 wins in 185, but 22-17 allows White 4 additional plies. Cumulative slip = 180 plies. The position here at move 100 is the same as was seen at move 66, playing in the cycle from 68 plies ago. See note [46].
63. 10-06 wins in 185, but 10-07 allows White 4 additional plies. Cumulative slip = 184 plies. The position here at move 102 is the same as was seen at move 68, playing in the cycle from 68 plies ago. See note [47].

SEARCH AND KNOWLEDGE IN LINES OF ACTION

D. Billings and Y. Björnsson

Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada

{darse,yngvi}@cs.ualberta.ca, http://www.cs.ualberta.ca/

Abstract This paper describes the design and development of two world-class *Lines of Action* game-playing programs: YL, a three time Computer Olympiad gold-medal winner, and MONA, which has dominated international e-mail correspondence play. The underlying design philosophy of the two programs is very different: the former emphasizes fast and efficient search, whereas the latter focuses on a sophisticated but relatively slow evaluation of each board position. In addition to providing a technical description of each program, we explore some long-standing questions on the trade-offs between search and knowledge. These experimental results confirm the conclusions made by earlier researchers in the domain of chess, thus showing that the trends are not game-specific. In particular, we see diminishing returns with additional search depth, and observe that the knowledge level of a program has a significant impact on the results of such experiments.

Keywords: Lines of Action, search, knowledge

1. Introduction

One of the most important considerations when designing a strategic game-playing program is the trade-off between knowledge and search.

To decide on the best move continuation, programs typically perform a look-ahead search, evaluate the positions at the leaves of the search tree, and then propagate those values back to the root using the *minimax* principle. A program that uses a sophisticated but time-consuming board evaluation can more accurately determine the merit of each game-state visited, at the cost of sacrificing some of the look-ahead depth. Conversely, a program that uses a faster but less sophisticated board evaluation method can perform a deeper search, improving its short-term tactical ability. There is also compensation toward better knowledge, in that each additional level of search provides a more refined approximation of the value of each preceding position.

The trade-off between knowledge vs. search has spurred a considerable amount of research interest in the past, mainly for the game of chess (Schaeffer, 1986; Berliner et al., 1990; Junghanns and Schaeffer, 1997; Heinz, 2000). This

paper provides further insights using the game of *Lines of Action* (LoA for short) as a new test-bed. LoA is tactically and strategically complex, and programs can employ many of the advanced search techniques and enhancements used by successful chess programs.

Two of the world's strongest LoA programs were developed hand-in-hand at the University of Alberta (Billings, 2000). One program, YL, developed by Yngvi Björnsson, uses a very fast but somewhat restricted framework for board evaluation, allowing deep look-ahead. The other program, MONA, developed by Darse Billings, employs a relatively slow evaluation function resulting in shallower search, but with added features that provide a better assessment of each position encountered. This fundamental difference in design philosophy provides an opportunity to investigate the relative importance of knowledge versus search.

The main contributions of this paper are: (a) descriptions of the design and cooperative development process of two high-performance game-playing programs, and (b) experimental investigation of some general trade-offs between search and knowledge, using a domain that is different from chess, but belongs to the same class of games.

The next section briefly presents the rules of LoA and summarizes important strategic game concepts to be considered by programs. Section 3 describes some of the many benefits of the co-development process. Sections 4 and 5 provide detailed technical descriptions of YL and MONA, respectively. Section 6 provides empirical results and some knowledge versus search experiments using the two programs. Finally, Section 7 summarizes the content and states conclusions.

2. Lines of Action

Lines of Action was invented by Claude Soucie in the early 1960s, and was popularized by Sid Sackson in his book "A Gamut of Games" (Sackson, 1969). The simple, elegant rules are now presented, along with an overview of some of the important strategic concepts that a high-performance program might consider incorporating.

2.1 Rules

Objective. The object of the game is to move all of your pieces into a single connected group. Pieces may be connected diagonally or orthogonally. The leftmost diagram in Figure 1 shows the initial board layout.

Movement. Black moves first, and players alternate, moving one piece per turn. A piece may move horizontally, vertically, or diagonally. Along a given line, the distance a piece moves is the same as the total number of pieces (of both colours) on that line. You may jump over your own pieces, but not your

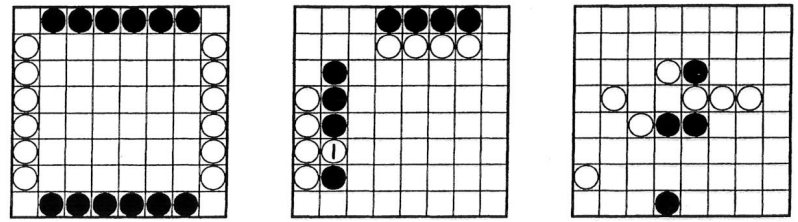

Figure 1. Initial LoA board layout, blockades, and mate-threat examples.

opponent's pieces. You may land on and capture your opponent's pieces, which are then removed from the game. You may not land on your own pieces.

2.2 Strategic Concepts

In chess and checkers, having more pieces than the opponent is highly correlated with winning, and this property outweighs all other factors in importance. In contrast, there is no single dominant feature in the assessment of LoA positions. As such, it is quite common to make concessions in one positional factor in order to strengthen another, and strong programs are frequently able to manage these trade-offs in order to maximize several features simultaneously.

We now list some of the important principles of LoA encountered during the development of MONA and YL.

Material. In LoA, there is no clear consensus on whether having extra material is advantageous, neutral, or detrimental. Since the goal of the game is to connect all of ones pieces into a single group, having fewer pieces can require less work to fully coordinate them. In contrast, having more pieces might make it easier to form one large group, and might also enable better control of the board, preventing the opponent from connecting their pieces. It may be the case that having more pieces than the opponent only offers an *indirect* advantage, by increasing the value of other properties mentioned in this section. Those indirect advantages may exceed the added liability of managing the extra pieces, yielding a net positive effect for material advantage. However, since those other attributes are being measured separately, the weight assigned to material difference may be zero, or negative.

Mobility. As with many other board games, it is normally advantageous to have a position with many options and possible continuations. Increased mobility generally entails increased flexibility. Simply having many moves available can make it easier for a player to develop their own plans, interfere with the opponent's plans, and defend against an opponent's immediate threats. Moreover, several types of moves can be identified, such as: moves that capture a piece, moves toward or away from the center of the board, moves that connect our pieces, or moves that cut an opponent group. Each of the distinct move types

can then be evaluated differently. Having **the move** (i.e., being the next player to move in a given position) can also be treated as a distinct characteristic of the position. The value of this privilege depends on other positional properties, and generally increases toward the end of the game.

Centrality. In LoA, controlling the center of the board is very important. This can be accomplished by direct occupation of the more central squares, or by tactical counter-measures that prevent the opponent from occupying those squares. The center is particularly important in view of the standard starting position. Since each side must unite pieces from opposite sides of the board, seizing control of the center gains the shortest route to unification, while simultaneously interfering with the opponent's connection. Having a bias toward centrality also has added pragmatic value, giving the program a sound high-level "plan" of bringing its pieces together in the middle of the board.

Piece Coordination. There are several identifiable concepts under the broad heading of piece coordination. Each of these is normally with respect to the pieces of the same colour. First, we say two pieces are **connected** if they are orthogonally or diagonally adjacent to each other. A **group** is a strongly connected subset of pieces (the object of the game being to form a single group). A program may designate a **main group** to be the largest group, or perhaps the most central group. The concept of **connectivity** can be measured as the number of pairwise connections between pieces; or the total number of groups; or the number of pieces that are not connected to the main group. The **proximity** or **cohesion** is a measure of distance or scatteredness of a player's pieces. An **outlier** is an isolated piece (typically at the edge of the board) that needs to be brought into connection with or proximity of the main group, or the majority of like-coloured pieces.

Obstructions. An opponent's piece or group of pieces may constitute an **obstruction** to connection. A piece may **block** one direction of movement of an enemy piece. A strong defensive formation is a **blockade** along the second rank or file, which greatly restricts the mobility of enemy pieces along the edge, and disconnects them from other like-coloured pieces. The effectiveness of an enemy blockade can be greatly reduced by having a **foothold**, which is a piece on the second rank that extends the edge group toward the center, and creates a defect in the blockade wall. The center diagram in Figure 1 shows a strong blockade for White along the top edge, and a White foothold (labeled '1') in Black's blockade on the left.

Mate Threats. A **mate threat** is a threat to win the game on the next move, by connecting all pieces into one group. In general, mate threats are devastating in LoA, since the opponent typically must weaken their position considerably to answer the threat. Given the rather highly constrained movement options, this

will commonly lead to subsequent mate threats, until finally there is no adequate response. The rightmost diagram in Figure 1 shows an example position where a long sequence of mate threats secures Black a win. Since they frequently lead to forced winning sequences, it can be worthwhile to detect statically certain types of mate threats, and give a large evaluation bonus for each one present. This property of LoA also encourages special-purpose algorithms near the end of the game, such as *threat-based search*; or the *proof-number* techniques seen in Sakuta et al. (2002) and Winands, Uiterwijk, and Van den Herik (2002).

3. Co-development, Co-evolution

The development of both YL and MONA began in February of 2000. Since YL was expected to be clearly superior in terms of engineering and search speed, the author of MONA decided early on to focus on having superior knowledge in the form of a better-informed evaluation function. This turned out to be a fortuitous decision, as the contrasting styles of play enabled both sides to learn far more from friendly contests than would have otherwise been possible, and the progress of both programs was greatly accelerated as a result.

YL is based on a very fast framework, and its evaluation function is fully incremental, meaning that it has very little work to do at each leaf node. In contrast, MONA has a work-intensive evaluation function applied to each leaf node. Overall, YL is about 22 times faster than MONA in terms of positions processed per second. In compensation, MONA evaluates each position somewhat more thoroughly.

To build a high-performance search engine like YL, the philosophy is: "start fast and stay fast", meaning that speed considerations and optimizations must be made at every stage of development. However, it can become increasingly difficult to make significant changes to the highly constrained architecture. In contrast, the design of MONA is basic and flexible, so new features can be added without difficulty. Since the evaluation function is already very costly, new attributes can be added that are rather expensive to compute, with only a minimal impact on overall speed performance. One might say "if you're slow anyway, take advantage of it!".

These fundamental differences in approach significantly enhanced the co-evolution of YL and MONA. A much broader range of positions were explored than would been seen with self-play matches, and critical weaknesses in each program were quickly revealed when playing into the other program's strength. The advantages of cooperative development do not end there. Since evaluation features are easy to add and experiment with in MONA, the slower program could be used as a proving ground for new ideas. If certain properties prove to be extremely valuable, they could then warrant the more difficult changes in YL. One example of this cross-fertilization occurred with "footholds" (a piece

on the second rank that diminishes the effect of an opponent blockade, as shown in Figure 1, and discussed in Section 5). This feature turned out to be so valuable in practice that a special effort was made to detect similar patterns within the framework of YL's fast evaluation function. The co-evolution worked in both directions. For example, games that MONA lost to YL due to short-term tactical errors suggested game-specific knowledge that could be added to reduce the risk associated with that weakness. As a result, some of MONA's knowledge is designed to compensate *directly* for the shallower search.

The development of MONA and YL was greatly facilitated by two e-mail games played against Kerry Handscomb (one of the strongest LoA players in the world, having tied for first place in the 2000 e-mail championship). Early versions of MONA and YL combined efforts against him, choosing the move with highest average score. The lessons learned from those games, and Kerry's commentary, resulted in major improvements to the evaluation functions of both programs. Kerry also wrote a series of informative articles on LoA for the magazine *Abstract Games* (Handscomb, 2003) (see issues 1–3, and others). Another valuable source of LoA domain knowledge was Dave Dyer's (2003) excellent website for the game, which includes the game records of past e-mail championships. For other resources, see the MONA and YL webpage (Billings, 2000).

4. YL

This section describes the architecture of YL, including the underlying framework, the board-evaluation scheme, and the search algorithm.

4.1 Line Decomposition

The program evaluates board positions line by line — that is, each file, rank, and diagonal is evaluated independently. The score of a board position is the sum of the scores of its lines. The board is decomposed into 32 lines as shown in Figure 2. The first diagram pictures the 8 files, the second diagram the 8 ranks. The two remaining diagrams show the diagonals, which are paired to form 8-square-long diagonals, hereafter referred to as *extended diagonals*. This pairing is done to achieve a more compact representation. An extended diagonal is still considered as two distinct diagonals for evaluation purposes.

Evaluating the lines independently makes it possible to use a fast table look-up evaluation scheme to score the board during game play. For each line there are only $3^8 = 6561$ possible different piece configurations. The total number of configurations (32×6561) is thus small enough to be evaluated beforehand. This evaluation is done at program startup and stored in tables residing in memory, called *evaluation tables*. The game is divided into three game phases

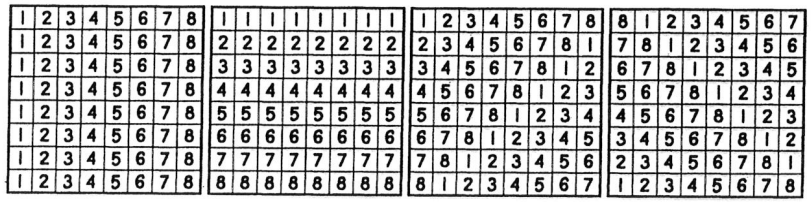

Figure 2. Example lines: file, rank, and extended diagonals.

(beginning, middle, and endgame) and different evaluation tables are used for each phase.

The program represents a board position internally using integers, one for each line. Each integer takes a value in the range 0 to 6560, representing the current piece configuration for the line. Piece configurations are mapped into integers as:

$$s_8 \times 3^7 + s_7 \times 3^6 + \ldots + s_2 \times 3^1 + s_1 \times 3^0$$

where s_i identifies the occupant of the i-th square on the line ($empty = 0$, $black = 1$, and $white = 2$). These piece-configuration numbers are updated incrementally as moves are made on the board. Removing or adding a piece from/to a square affects only the configuration of the four lines intersecting the affected square. One benefit of this representation is that the piece-configuration numbers can be used directly as indicies into the evaluation tables. Evaluating a board position is then simply a matter of looking up the merit of each line in the evaluation table. This evaluation is also done incrementally by keeping track of the current board score, and adjusting it by the evaluation differences of only the line configurations that changed during a move. This requires only a few table lookups. This efficient way of representing and evaluating boards is not new. Similar board representations are used by some high-performance Othello programs (Buro, 1999).

4.2 Evaluation Function

The main attraction of the aforementioned line-by-line evaluation scheme is its efficiency. On the down-side, the type of features that can be expressed within this framework are necessarily somewhat restricted. However, several important features can be measured precisely (including material balance, number of connections, and piece mobility), whereas other features must be approximated (such as proximity, obstruction, and blockage effectiveness). Where such approximations are not sufficient we use special non-line-based patterns.

Material. YL has a slight dislike for being up material, increasing as the game progresses. Note that this does not necessarily imply that it is bad to capture pieces. Rather, YL captures pieces only if it gives positional advantages.

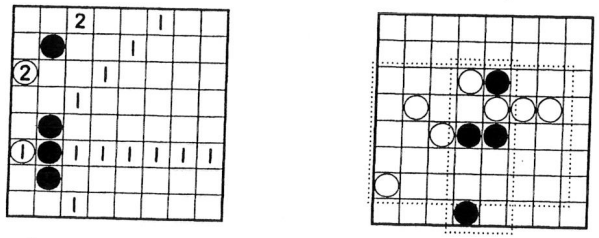

Figure 3. Example of a blockage penalty, and proximity bounding boxes.

Mobility. The program distinguishes between four types of moves (in decreasing order of importance): capture moves, moves that establish a connection, regular moves, and moves that disconnect own piece formation. Note that a single move can belong to more than one category. The merit of such a move is the combined merit from all relevant categories. Also, the side to move gets a constant bonus.

Centrality. A bonus is given for both direct and indirect center control; the more centralized a piece is the higher bonus it gets. Pieces sitting on the edge of the board are penalized, although somewhat less if they can move towards the center.

Piece coordination. YL measures connectivity by summing the number of neighbours of the same colour over all pieces on the board. It also measures line-wise piece proximity (how far apart pieces lie on a line). However, experience showed this measure to be insufficient on its own. Therefore, before the Computer Olympiad in 2002, a new non-line-based proximity feature was added. It keeps track of the area (number of squares) of the minimal bounding box needed to enclose pieces of each side; the smaller the box, the higher bonus a side gets. These boxes also encourage outliers to start gravitating toward the rest of the pieces. In Figure 3, the diagram on the right shows the bounding boxes for both sides.

The program has no notion of how many groups there are on the board except, of course, detecting the "single group" end-of-game condition. A single remaining group is detected by doing a breadth-first search over all neighbours of the same colour, starting with the piece last moved. If the number of pieces visited is equal to the number of pieces a player has, we know the pieces form a single group (in case of a capture move we need also to check end-of-game condition for the opposing side). Detecting the single-group condition initially slowed the program down significantly. However, by keeping two bitmasks for each side indicating which columns and rows its pieces occupy, we can cheaply check for necessary conditions of a single group being formed, in which case we then do the more expensive connection test. In practice, this trick eliminates most calls to the breadth-first search. These bitmasks are also used to efficiently keep track of the proximity bounding boxes.

Obstructions. YL gives additional penalty to edge pieces that are fully or partially blocked by the opponent's pieces. For example, in the diagram on the left in Figure 3 both white pieces are penalized. The number of penalty points is determined by the length of the blocked lines: piece 1 gets in total 11 penalty points (shown with an '1') whereas piece 2 is only penalized by only 1 point (shown with a '2'). However, this scheme overestimates the penalty for blockades with footholds (the blockade is less effective because the edge pieces are connected to the outside via the foothold piece). A line-by-line evaluation scheme is unable to detect such situations. Footholds do occur frequently enough in practice to warrant a special treatment. Thus a special configuration pattern is used in YL that looks at the second file/rank in conjunction with the first, allowing the program to detect whether blocked edge pieces are connected to the outside and then scale down the blockade penalty appropriately.

YL also detects (along lines) how many opposite coloured pieces there are in between one own pieces, slightly penalizing such obstructions.

4.3 Evaluation Weights

Each line is evaluated using the aforementioned features that are then combined into a single line value using a linear function. A linear function was chosen somewhat arbitrary. In practice we could equally well have used a more complex non-linear function without sacrificing performance. However, we have not experimented with such alternatives.

Instead of hand-tuning the evaluation weights, we initially used a temporal-difference learning method for determining the relative importance of each evaluation feature. This was a sensible decision given the limited expert knowledge about the game, and allowed us to obtain a initial set of weights superior to what we could come up with by hand. A version of the program using the learned weights won the gold-medal at the Computer Olympiad held in London in 2000. However, further tuning of the weights (mainly based on observations from tournament play) and, in particular, introduction of new evaluation features have since then significantly increased the program's playing strength.

4.4 Search

YL uses a traditional alpha-beta-based search algorithm (more specifically *Principal Variation Search* (Marsland, 1982)). The algorithm is augmented with many state-of-the-art enhancements, such as: iterative deepening, aspiration windows, a two-level transposition table, extensive automatically built opening book, repetition detection, and thinking on opponent's time. The program also employs two well-documented speculative pruning schemes: null-move (Beal, 1989) and multi-cut pruning (Björnsson and Marsland, 2001).

As mentioned earlier, evaluation of game positions is done incrementally using table lookups. Move generation is also relatively fast, in part because legal moves for all possible line configurations are pre-calculated at program startup. The program employs both static (based on evaluation-table values) and dynamic (transposition-table move, killer-moves and history-heuristic) move-ordering techniques. A hierarchical move-ordering approach is used: in the upper levels of the tree (closer to the root) a more sophisticated move ordering is employed whereas at the lower levels a faster, although somewhat less sophisticated, ordering mechanism is used. All together, this results in a very fast and efficient search (the program typically explores close to 1.5 million positions-per-second in the opening on a 2.4GHz P4 PC).

5. Mona

This section describes the search engine and the evaluation function of MONA. As stated before, emphasis is on evaluation.

5.1 Search Engine Components

MONA is a fairly basic alpha-beta search program, using Principle Variation Search. The data structures for board representation, evaluation features, and move lists are simple integer arrays. Most of the well-established search enhancements are used, such as iterative deepening, transposition tables (with embellished Zobrist hashing), and the null-move heuristic.

Since *move generation* is used in the leaf-level evaluation function as well as the search process, the program spends a significant fraction of its execution time in this procedure. An optimization called *move gathering* was implemented, where all possible moves for each line configuration (row, column, or diagonal) are pre-computed, and those short move lists are concatenated at runtime. This resulted in a 40% speed-up to the program. A further 200-300% speed-up might be possible by incrementally carrying the move list indices (the new bottleneck), but this was not done prior to the program's retirement in 2001.

Good *move ordering* is accomplished with a two-level hierarchy. First, the *transposition table move* is considered (i.e., the move which produced the best score the previous time the position was encountered, typically in the previous iteration). The second level is the default *static move ordering*, which ranks the general desirability of each move. A move toward the center of the board is rated higher (specifically, the centrality of the destination square), and capture moves are given a small bonus. Since this ranking is over a fixed interval (2-16), a *linear time Radix sort* is used to order the move list. This default move ranking is built into the move generator to reduce overhead. Move ordering with killer moves and the history heuristic are available as an option, but do not significantly increase program performance.

Since a prominent *odd-even* effect is observed in evaluations, and since mate threat detection (described below) only occurs on odd-ply searches, MONA iterates 2-ply at a time. The courser granularity of iterative deepening results in an inefficient use of time when using a fixed-time-per-move time control, but this is partially offset by the savings from not doing even-numbered iterations.

5.2 Evaluation Function Components

The strength of MONA lies in the evaluation function, which attempts to assess several properties of strategic importance, in the hope that this information will more than compensate for the overhead added by the relatively expensive computations.

The most basic evaluation simply determines whether the position is won, lost, drawn (by simultaneous connection), or unknown. This is done with a low-overhead breadth-first search to identify each group. If there is only one group of a given colour, then the game is over. The most important components of the full evaluation function are *centrality*, *mobility*, *thickness*, and *mate threats*. Useful refinements consider *outlier mobility*, *blockades*, *footholds*, *outlier blocking*, the *progress* toward connectivity, and the value of *the move*.

In most cases, it is the *net difference* between Black pieces and White pieces that is of interest. For example, the program would willingly reduce its own mobility provided that the opponent's mobility is reduced by an even greater amount. For the most part the evaluation is symmetric (with the exception of outlier mobility, and mate threats).

Centrality. From a practical programming point of view, centrality is more than just a feature – it can be thought of as an overall game plan. The other two most important features in MONA's evaluation function (mobility and thickness) also have a significant centrality bias. This lends a degree of "harmony" to the evaluation, in that they are all striving for mutually supportive goals, rather than being at odds with each other. And indeed, it is quite common to sacrifice some of one commodity in order to gain more of another, in a cyclic process that eventually reaches positions that are powerful on all three counts.

To quantify centrality, each square is assigned a weight corresponding to the sum of orthogonal distances from the nearest corner (the four corner squares having a weight of 2, up to the four center squares having a weight of 8). The net centrality is *relative* to the number of pieces remaining. Thus, a few pieces on squares near the center would have a higher average centrality than a large number of pieces scattered about the board.

Mobility. A basic measure of mobility would simply count the number of moves that can be made. However, some moves are generally better than others. As noted above, the static move ordering value is determined by the destination centrality, and whether the move captures an enemy piece. By summing over

those values, the mobility function is naturally biased toward the moves that are likely to be *useful*. This is an *absolute* measure, so having more pieces, and thus more moves, is generally favourable.

An interesting consequence results from giving capture moves a greater weight. This actually *discourages* even exchanges, because having the *option* to make a capture has more value than actually making that capture. Thus, all else being equal, the program favours building up the pressure on key squares rather than releasing the tension through an even exchange. This has considerable practical value, especially in play against humans, because the computer program can handle the extra burden of many complex continuations much better than a human player. The chance of the opponent making a fatal error is thus increased in practice. This principle is highly analogous to the famous chess adage "the threat is stronger than the execution".

Thickness. In general, "clumping" of pieces is desirable, and there is some value in having redundant connections, preventing a group from being cut into two. However, we want to avoid having groups that are "too heavy", thereby reducing its own mobility.

The measure of connectivity used by MONA is called *center-thickness*, or simply thickness. A straight-forward measure would count the number of pairwise adjacent pieces on the board. The embellishment used is to weight each of those pairs according to the centrality of the squares they occupy. This is another cumulative measure, so having more pieces is generally favourable. MONA uses a zero weight for the *material* factor, but still exhibits a preference for extra pieces, based on mobility and thickness.

It should be clear that the centrality biases built into mobility and thickness will generally encourage pieces to be moved away from the edge; and for groups to be formed in the center, if possible. However, this is not a heavy-handed bias, and cannot be easily obtained against quality opposition. It is simply a preference over other types of moves and piece formations. The relative weights of these three evaluation terms were set to have roughly equal contributions, with a slight preference for mobility, since it usually has a bit more pragmatic value. Very little was done in the way of tuning, and it is unknown if the program's performance could be enhanced significantly with more thorough experimentation.

Mate Threats. MONA computes many useful properties for each position. All groups are identified with the breadth-first search described previously. MONA designates the largest group to be the *main group* (choosing arbitrarily among equals).

Since the full move list is also available, it is possible to selectively do a check for the case where a single remaining outlier has a move that will put it adjacent to the main group, which constitutes an immediate threat to win the game.

This particular type of mate threat is the most common in practice, and can be detected without a full extra ply of look-ahead. While it is possible to write a special-purpose mate solver that looks only at direct threats and responses, we found that most of that utility was accomplished by simply knowing that a threat exists. MONA assigns a huge bonus for each such threat, which dominates all other evaluation terms. Thus it will always choose the best move among those that contain a threat (or highest number of multiple threats).

This policy is something of a gamble, since the threat may not actually lead to a win. However, to date we have only seen two cases where a winning position was lost by chasing a specious mate threat (both were against YL and, unfortunately, one in an important game).

Outlier Mobility. Given the rich information maintained about the board position, MONA can determine the individual mobility for every piece that is not part of the main group. She applies a non-linear penalty to the least mobile outlier for each side. Only the single worst outlier is considered. The search will naturally uncover combinations of moves that improve more than one outlier.

The weights for this feature are heavily skewed, making our worst outlier more important than the opponent's worst outlier. The reasoning is that we do not want to invest a lot of energy (and evaluation points) on trying to trap an opponent piece that might easily escape, leaving us in a weakened position. Conversely, it is also risky to allow our own outliers to be trapped, since there may be no effective way to solve the problem (especially against humans, who can easily visualize such futures). The program is careful to avoid losing a game due to such traps, while preferring to build up steadily a strong position rather than trying to trap the opponent. This is one of several examples where the evaluation is consistent with the natural strengths and weaknesses of the program.

Blockades and Footholds. MONA's evaluation function expends a lot of effort on the analysis of blockades along the second rank (see center Figure 1). These formations arise naturally from the standard starting position, even when using only the basic evaluation knowledge. The special purpose evaluation assesses the effectiveness of each blockade. Each additional blocking piece has a multiplicative effect on the penalty, while the presence of a foothold nullifies it almost completely. The program also distinguishes between a blockade of the main group and a blockade of outliers, with the latter being more serious.

Other Features. Since the character of LoA positions change radically during the course of the game, it is desirable to alter the overall plan and assessment to match the prevailing conditions. As a case in point, even the most refined positional evaluation is of little use in the final stage of the game – the only relevant question is whether we can form a single connected group before the opponent does. Empirically, it was found that the value of having *the move*

increases steadily toward the end of the game, when having the initiative is usually decisive.

The *progress* toward game completion is actually measured in a variety of ways. One of the simplest is to count the number of remaining pieces that are not part of the main group. A bonus is given for having two remaining outliers, and a larger bonus for having only one (since mate threats are commonly on the horizon). However, it is dangerous to try to converge too quickly, as it may fail tactically after all of the positional advantage has been sacrificed.

Many of the strategic properties perceived by MONA's evaluation function cannot possibly be uncovered within a practical search horizon. For example, a piece trapped behind a wall of enemy pieces can be identified by static analysis, but the consequences of having that piece trapped might not begin to be felt until many moves in the future. MONA can add this type of knowledge without much down-side, whereas it is difficult to define within the framework of YL, and might be prohibitively expensive in any case (resulting in a net decrease in performance).

6. Empirical Results and Experiments

In this section we first provide insights into the playing strength of YL and MONA by reviewing tournament results. Secondly, we investigate the trade-offs of knowledge versus search in the game of LoA both via self-play and by matching the two programs against each other.

6.1 Over-The-Board Competitions

After a series of mutually beneficial friendly matches against each other, YL and MONA made their competitive debut in the University of Alberta Lines of Action Open, in April 2000 (Billings, 2002). The tournament was a double round-robin format with a time control of 20 seconds per move. YL won with a perfect 22-0 score, and MONA finished second at 19-3.

In August 2000, both programs competed in the Fifth Computer Olympiad in London, England. YL won the gold medal, and MONA won the silver. Although MONA lost a critical game to YL on time only seconds before proving a win, both authors believed YL to be the stronger program at the 30-minute per game time constraints. The program MIA, by Mark Winands at University of Maastricht, took the bronze medal. YL successfully defended its title at the Sixth Computer Olympiad in 2001 ahead of MIA-II, and again won the gold medal at the Seventh Computer Olympiad in 2002 ahead of a steadily improving MIA-III (Björnsson and Winands, 2002). MONA did not compete in either event. In July 2002, a four game friendly match was played between MIA-III and the original MONA from 2000. Each program won two games, and based on further analysis of the moves played, it appeared that MIA-III had largely closed the gap that previously existed.

Rank	Rating	Colour	Name (Country)
1	2763		MONA (Canada) 25 wins, 0 draws, 0 losses
3	2374	W	Jorge Gomez Arrausi (Spain)
4	2202	WB	Claude Chaunier (France)
5	2192	BW	Kerry Handscomb (Canada)
6	2102	W	Uli Vogel (Germany)
7	2086	W	Ragnar Wikman (Finland)
8	2062	W	Hartmut Thordsen (Germany)
10	1999	W	Dave Dyer (USA)
11	1981	BB	Patrick Duff (USA)
13	1919	B	John Bosley (New Zealand)
18	1871	W	Fred Kok (Netherlands)

Table 1. MONA's e-mail results against the top human players.

6.2 E-mail Correspondence Competitions

At deeper search depths, MONA's strength increases dramatically. However, due to the expensive evaluation function, it can take a few hours to complete 11-ply early in the game, or 13-ply in the middlegame. This makes MONA particularly well-suited to e-mail correspondence play, with a pace of roughly one move per day.

Beginning in the summer of 2000, MONA began playing on Richard's PBeM server (a popular play-by-email service) against many of the strongest known human players, winning every game played. MONA then won the Fifth Annual E-mail Tournament (the *de facto* world championship) with a perfect 14-0 record, including wins over most of the best LoA players in the world.

Table 1 lists some of the e-mail games played by MONA from May 2000 to May 2001. The chess-style ratings were calculated independently, using iterative re-computation over a database of more than 1000 PBeM LoA games until reaching convergence. The #2 rated correspondence player, at 2417, is the program MIA, which has only lost to the top human player, Jorge Gomez Arrausi. Jorge Gomez Arrausi won the 2000 e-mail championship, and was the top finishing human again in 2001, losing only to MONA. Several of the players listed are former LoA medalists at the Mind Sports Olympiad, including Fred Kok (gold twice), Hartmut Thordsen (gold), Ragnar Wikman (silver twice), and John Bosley (bronze).

MONA had the second move in most of the games against the top players, which is believed to be a larger disadvantage than having the Black pieces in chess. MONA also used considerably less time than her human opponents. In the final round of the 2001 e-mail tournament, MONA used an average elapsed time of 3.2 days per game, while her opposition used an average of 42.7 days per game against her. Based on the perfect record against elite competition, it is safe to conclude that the playing strength of MONA exceeds that of all human players by a considerable margin. However, it should be noted that LoA is still a young game, growing in popularity, and it is possible that "grandmaster"

calibre players could emerge in the future, giving programs a tougher challenge than has been seen to date. Programs will also continue to improve, and as they help humans to deepen their understanding of the game, that in turn could provide new knowledge to be added to future programs.

In April 2001, an 8-game match was played between older versions of MONA and YL, using the correspondence time control of 8 hours per move. Each game took roughly one week to complete. It is likely that these games constituted the highest level of play ever attained in LoA at that time. MONA won the match convincingly, with 7 wins and 1 loss. We intend to repeat this experiment using a more recent (and much stronger) version of YL.

6.3 Knowledge versus Search Experiments

In general, the farther a program looks ahead, the better it plays. This is the main justification for designing fast-searching game-playing programs. Historically, deeper search in chess programs has always led to significant improvements in performance. However, experimental studies have demonstrated diminishing returns with additional search depth (Junghanns and Schaeffer, 1997; Heinz, 2001).

We are also interested in investigating the importance of knowledge as LoA programs are given more time to think, since this will be a good predictor of how faster hardware platforms in the future will affect a program's playing strength. Traditionally, such investigations have involved a series of self-play experiments.

Constant Knowledge (Self-play) Experiments. First we repeated the most common self-play experiments, using YL and MONA. The results of those matches are shown on the left in Table 2. Each data point is the outcome of a 200-game match. A standardized set of 100 3-ply openings was defined (available upon request), and each player played both sides of each opening.

As in chess, searching deeply is obviously important: the deeper searching program invariably outperforms the shallower searching program by a considerable margin. However, as the search depth increases, the winning margin decreases, supporting the aforementioned experimental results found in the domain of chess.

Also of interest is that YL appears to benefit more from the deeper search than MONA. As noted in previous discussion, some of the knowledge in MONA

Time(sec)	YL vs YL01	YL vs MONA
2	77.50	81.00
8	79.75	79.25
32	83.25	80.25
128	81.00	65.75

Depth	YL vs YL	MONA vs MONA
5 vs 7	89.75	79.50
7 vs 9	85.75	78.00
9 vs 11	79.75	72.50
11 vs 13	79.00	-
13 vs 15	72.75	-

Table 2. Fixed and variable knowledge experiments.

directly compensates for the shallower search; whereas YL must depend on the deeper search to actually witness certain short-term tactics, refining its minimax evaluation of the position. The results of 200-game matches alone are not conclusive, but the same trend has been observed for other experiments as well.

Varying Knowledge Experiments. The self-play experimental setup only shows the benefits of additional search when playing against *a very similar* program. This does not address the possible effects of knowledge. The experiments reported above might be misinterpreted to infer that YL would benefit more than MONA from faster hardware – the exact opposite is true!

To test the knowledge differences directly, YL was also played against the 2001 version, labeled YL01. The more recent version was greatly improved with the addition of several types of knowledge and evaluation features, but the two YL programs are almost identical in search capacity, due to the precalculated evaluation tables described earlier. Although MONA has not undergone any significant changes since the 2000 Computer Olympiad, it is still regarded to have the best-informed evaluation function of the three programs.

The results of those matches are shown to the right in Table 2. At shorter time controls, YL outperforms both YL01 and MONA by a similar winning margin. As the time controls get longer, YL continues to outperform YL01 at a comparable win rate, indicating that the improvements in knowledge (the only significant difference between the two programs) continue to pay dividends as search depth increases.

However, the win rate against MONA drops off dramatically once the latter is given at least two minutes per move, despite the fact that YL continues to outsearch MONA by almost 3-ply on average. (This trend continues at longer time controls, but those experiments were not complete and are not shown here). Presumably, the fast search engine is gaining less and less from deeper search, while the knowledge advantage continues to provide sustainable benefits.

Further evidence is seen in matches between MONA and YL with equal fixed depths (using comparable sets of search enhancements). Whereas MONA won about 55% of games at 5-ply, 7-ply, and 9-ply (54.50%, 55.25%, and 56.00%, respectively), her win rate increased to 71.00% when searching 11-ply per move.

The implication of these observations is that faster hardware platforms do indeed benefit knowledge-rich programs more than fast searchers. This in turn suggests that when developing strategic game-playing programs, time invested on improving the program's board evaluation will generally pay greater dividends in the long run than effort spent on search improvements (especially in view of the ever increasing difficulty in obtaining significant improvements in search efficiency). Similar behaviour has been observed in chess programs (Berliner et al., 1990).

7. Conclusions

We have revisited some long-standing questions regarding the roles of search and knowledge in high-performance game-playing programs, using two champion Lines of Action programs which emphasize different aspects of these contrasting approaches. Although the experiments are far from exhaustive, the results obtained so far are entirely consistent with previous studies for the game of chess. This supports the view that these are general phenomena, rather than game-specific.

By considering the effects of the knowledge level of game-playing programs over increasing search depths, it is possible to get a glimpse of what will likely be seen in the future. Although it is clear that search depth is and will continue to be very important, there are definite indications that increasing the knowledge holds the greater promise for lasting improvements in performance.

References

Beal, D. (1989). Experiments with the null move. In *Advances in Computer Chess 5*, pages 65–89. Elsevier Science Publishers B.V. D. Beal (editor).

Berliner, H., Goetsch, G., Campbell, M. S., and Ebeling, C. (1990). Measuring the performance potential of chess programs. *Artificial Intelligence*, 43(1):7–20.

Billings, D. (2000). `http://www.cs.ualberta.ca/~games/LOA/`.

Björnsson, Y. and Marsland, T. (2001). Multi-cut alpha-beta pruning in game-tree search. *Theoretical Computer Science*, 252:177–196.

Björnsson, Y. and Winands, M. (2002). YL wins Lines of Action tournament. *ICGA Journal*, 25(3):185–186. See also: 23(3):179–179, and 24(3):180–181.

Buro, M. (1999). From simple features to sophisticated evaluation functions. In *Proceedings of The 1st International Conference on Computers and Games (CG '98). LNCS special issue Computers and Games*, pages 126–145. Springer-Verlag, Berlin, Germany.

Dyer, D. (2003). Lines of action. `http://www.andromeda.com/people/ddyer/loa/`.

Handscomb, K. (2003). Abstract Games. `http://www.abstractgamesmagazine.com`.

Heinz, E. (2000). *Scalable search in computer chess*. Vieweg Verlag, Germany.

Heinz, E. (2001). Self-play, deep search and diminishing returns. *ICCA Journal*, 24(2):75–79.

Junghanns, A. and Schaeffer, J. (1997). Search versus knowledge in game-playing programs revisited. In *IJCAI-97*, pages 692–697.

Marsland, T. A. (1982). Relative performance of the alpha-beta algorithm. *ICCA Journal*, 5(2):21–24.

Sackson, S. (1969). *A Gamut of Games*. Random House.

Sakuta, M., Hashimoto, T., Nagashima, J., and Iida, H. (2002). Endgame-search techniques developed in shogi: application to Lines of Action. In Caulfield, H. et al., editor, *Proceedings of JCIS 2002*, pages 458–460.

Schaeffer, J. (1986). *Experiments in search and knowledge*. Ph.D. thesis, Department of Computing Science, University of Waterloo, Canada.

Winands, M.H.M., Uiterwijk, J.W.H.M., and Van den Herik, H.J. (2002). PDS-PN: A new proof-number search algorithm: Application to Lines of Action. In *Proceedings of The 3rd International Conference on Computers and Games (CG '02)*. To appear.

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).

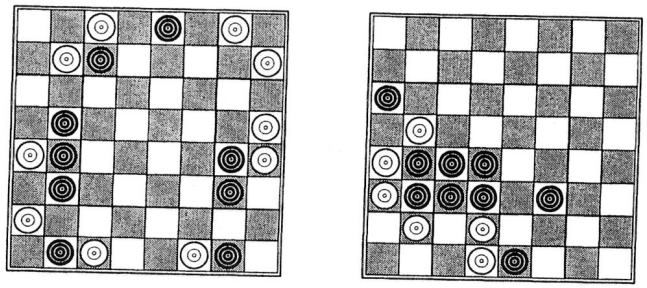

Figure 1. (a) Scattered Pieces (b) Position with two black Q_4's.

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.

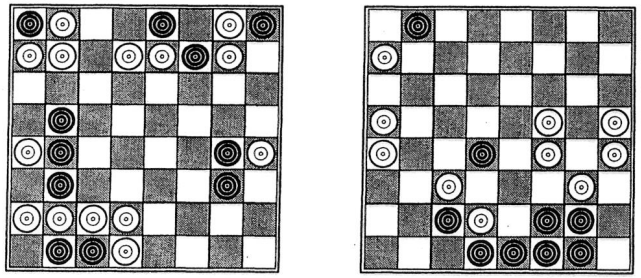

Figure 2. (a) Position with walls (b) Position with an outlier on **b8**.

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.