SEARCH AND KNOWLEDGE IN LINES OF ACTION

Darse Billings and Yngvi Björnsson
University of Alberta
{darse,yngvi}@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. The two contrasting approaches provide an excellent opportunity to explore some long-standing questions on the trade-offs between search and knowledge. We show that this domain is more suitable than chess for observing and measuring the effect of diminishing returns with additional search depth. We further show that the knowledge level of a program significantly affects the results of such experiments, which has not been considered in most previous self-play studies.

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. On the other hand, 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; Junghanns and Schaeffer, 1997; Heinz, 2000]. This paper provides further insights using the game of *Lines of Action* (LoA for short) as a test-bed. Like chess, LoA is both tactically and strategically complex. Programs for playing LoA can employ most of the advanced search techniques and enhancements used by successful chess programs. The average branching factor is also similar to chess. Thus, the observations and insights gathered for this domain are likely to be relevant to previous studies. Moreover, using other games with somewhat different properties helps to shed light on the general phenomena, rather than merely characteristics of a single specific game.

Two of the world's strongest LoA programs were developed hand-in-hand at the University of Alberta. One program, YL, developed by Yngvi Björnsson, uses a fast but somewhat restricted framework for board-evaluation, allowing very deep look-ahead. The other program, MONA, developed by Darse Billings, employs a relatively slow but sophisticated board-evaluation scheme, relying on the better-informed evaluation function to compensate for the shallower search. The fundamental difference in design philosophy of the two programs provides an excellent 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 of the trade-offs between search and knowledge, which benefit from certain properties of this domain, and from the contrasting styles of the programs.

The next section 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". 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.

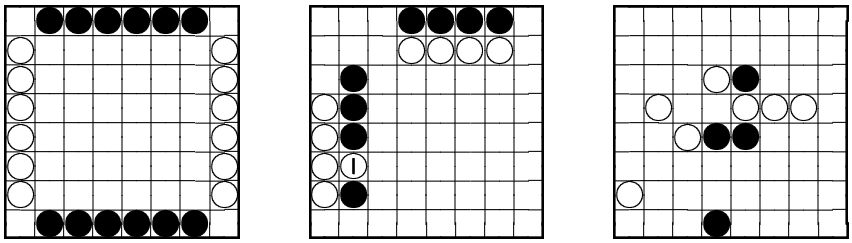

Figure 1. Initial LoA board layout, blockades, and mate-threat examples

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 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. On the

other hand, 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** (*ie.* 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 statically detect 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 *proof-number search*, as seen in top level Shogi programs [Iida et al., 2003], and more recently in the LoA program MIA [Winands, 2000].

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 *directly* compensate 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

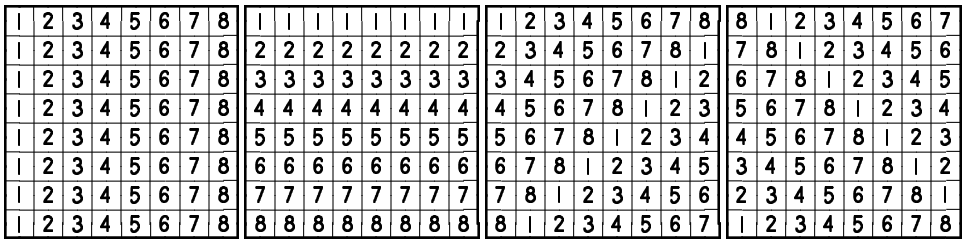

Figure 2. Example lines: file, rank, and extended diagonals

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]. Another valuable source of LoA domain knowledge was Dave Dyer's excellent website for the game, which includes the game records of past e-mail championships [Dyer, 2003].

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 (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:

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

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 ([Lee and Mahajan, 1990; 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 can 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.

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 neighbors of the same color 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 color, 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

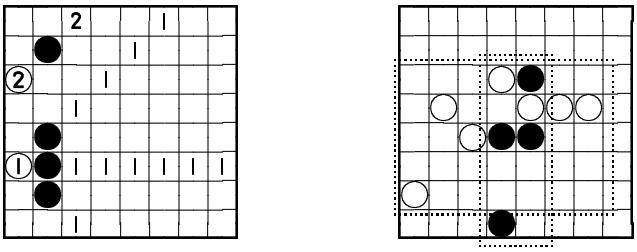

Figure 3. Example of a blockage penalty, and proximity bounding boxes

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 colored 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 hash table, extensive au-

tomatically 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 precalculated 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

5.1 Search Engine Components

MONA is a fairly basic alpha-beta search program, using a particularly clean formulation of 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 (*ie.* 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 for some of one commodity to be sacrificed for more of another, cycling through to eventually reach 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". This may have contributed to many of the easy wins MONA achieved as the second player, who begins the game with a substantial disadvantage.

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*.

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 (unfortunately, both were against YL, and 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 steadily build up 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

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, 2003]. 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 [Björnsson, 2000]. YL successfully defended its title at the Sixth Computer Olympiad in 2001 ahead of MIA-II [Björnsson and Winands, 2001], 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 evaluations of the moves played, it appeared that MIA-III had closed the gap that previously existed.

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 4 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

Table 4. MONA's e-mail results against the top human players

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)

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.

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, improved search in chess programs has always

led to significant improvements in performance. However, the general belief is that the benefits of additional search will eventually diminish as the search depth increases. Several experiments have indicated that the pay-back for additional search depth is starting to exhibit diminishing returns [Heinz, 2001].

In particular, we are interested in investigating the importance of knowledge as game-playing 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 usual self-play experiments, using YL, and MONA. The results of those matches are shown on the left in Table 5. Each data point of an experiment is the outcome of a 200-game match. A standardized set of 100 3-ply openings was defined, 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 noticably. Apparently, the diminishing returns of additional search depth occurs much sooner in LoA than in chess, making it easier to observe and quantify within comfortably tractable real-time searches.

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 directly compensates for the shallower search; whereas YL must depend on the deeper search to actually witness short-term tactics, and refine its minimax evaluation of the position.

One possible explanation for the faster diminishing returns of added search is that LoA may not be as tactically complex as chess. It could be the case that the tactical tension in a typical LoA position is dissipated more easily, whereas chess positions may continue to be very sharp for a long time. Consequently, the majority of tactical pitfalls in LoA might be avoided with fairly modest search depths, and thus positional play would become increasingly important. Further evidence of this "bounded tactical horizon" in LoA was seen in fixed equal depth matches between MONA and YL. Whereas MONA won 55% of games at 5-ply, 7-ply, and 9-ply, her win rate suddenly increased to 71% when searching 11-ply per move.

Table 5. Fixed and Variable Knowledge Experiments

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	-

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

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 interpreted to mean that YL benefits more than MONA from additional search depth, but it would be incorrect to infer that faster hardware would benefit YL more – the exact opposite is true!

To test this directly, YL was played against MONA, and also against the 2001 version, labeled YL01. The newer version was greatly improved with the addition of several types of knowledge and evaluation features, but the two programs are almost identical in search capacity, due to the pre-calculated 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 5. 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, a more interesting thing to notice is that the win rate against MONA drops off dramatically once the latter is given two minutes per move, despite the fact that YL continues to outsearch MONA by almost 3-ply on average. Presumably, the fast search engine is gaining less and less from deeper search, while the information advantage continues to provide benefits.

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 off better in the long run than effort spent on search improvements.

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 it is clear that search is, and will continue to be very important, there are definite indications that 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).

Billings, D. (2003). `http://www.cs.ualberta.ca/~games/`.

Björnsson, Y. (2000). YL wins Lines of Action tournament. *ICCA Journal*, 23(3):178–179.

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. (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.

Buro, M. (1999). From simple features to sophisticated evaluation functions. In *Proceedings of The First International Conference on Computers and Games (CG '98). Lecture Notes in Computer Science, special issue Computers and Games*, pages 126–145. Springer Verlag.

Dyer, D. (2003). `http://www.andromeda.com/people/ddyer/loa/Programs.html`.

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.

Iida, H., Sakuta, M., and Rollason, J. (2003). Computer shogi. *Artificial Intelligence*, 134:121–144.

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

Lee, K. and Mahajan, S. (1990). The development of a world class Othello program. *Artificial Intelligence*, 43:21–36.

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

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

Winands, M. (2000). Analysis and implementation of Lines of Action. Master's thesis, Universiteit Maastricht, Maastricht, The Netherlands.