Learning in Lines of Action

Mark H.M. Winands Levente Kocsis
Jos W.H.M. Uiterwijk H. Jaap van den Herik

Department of Computer Science, Institute for Knowledge and Agent Technology, Universiteit Maastricht, The Netherlands
{m.winands, l.kocsis, uiterwijk, herik}@cs.unimaas.nl

Abstract

This paper investigates to what extent learning methods are beneficial for the Lines of Action tournament program MIA. We focus on two components of the program: (1) the evaluation function and (2) the move ordering. Using temporal difference learning the evaluation function was improved by tuning the weights. We found substantial improvements for three weights. The move ordering was enhanced by the Neural MoveMap (NMM) heuristic, which is based on learning. The two learning techniques improved both the playing quality and the speed of the program. Test results are given. The new evaluation function improved the program with a winning ratio of 1.68. The speed up of the NMM heuristic is 17 percent.

1 Introduction

The standard framework of the $\alpha\beta$ search with its enhancements offers a good start position for building a strong game-playing program. For Lines of Action (LOA) we built the game-playing program MIA (Maastricht in Action) by carefully composing its "handcrafted" evaluation function and implementing the $\alpha\beta$ variant PVS with iterative deepening, transposition tables, quiescence search, killer moves, history heuristic, etc [8]. MIA so equipped, came second on the 2001 Computer Olympiad [3]. Further improvement of the program is expected to be achieved mostly from fine-tuning the components mentioned. One approach is to fine-tune them by hand. However, the program is playing at such a high level that the effect of the changes in most of the components are beyond human understanding. The alternative approach is to use automatic tuning by various learning techniques. In this paper we focus on improving two components: the evaluation function and the move ordering. To improve the evaluation function we employed temporal difference learning. This learning method was first used for checkers by Samuel [12] and was essential for building the world-champion-level Backgammon program by Tesauro [15]. For move ordering similar learning techniques were designed recently, of which the Neural MoveMap heuristic [6] seems particularly promising.

The remainder of this paper is organised as follows. Section 2 explains the game of Lines of Action and describes the tournament program. Next, the temporal difference learning and the Neural MoveMap heuristic are explained in section 3. Subsequently, the results of using these learning algorithms are presented in section 4. Finally, in section 5 we present our conclusions.

2 Tournament program MIA

MIA (Maastricht In Action)[1] is a LOA-playing tournament program. It is written in Java. Below we describe some details of MIA. In subsection 2.1 we explain the rules of Lines of Action. We give an overview of the MIA's evaluation function in subsection 2.2. The search engine is briefly described in subsection 2.3.

2.1 Lines of Action

Lines of Action (LOA) [11] is a two-person zero-sum chess-like connection game with perfect information. It is played on an 8×8 board by two sides, Black and White. Each side has twelve pieces at its disposal. The starting position is given in figure 1a. 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 (see figure 1b). 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 (see figure 1c). In the case of simultaneous connection, the game is drawn. The connections within the unit may be either orthogonal or diagonal. If a position with the same player to move occurs for the third time, the game is drawn.

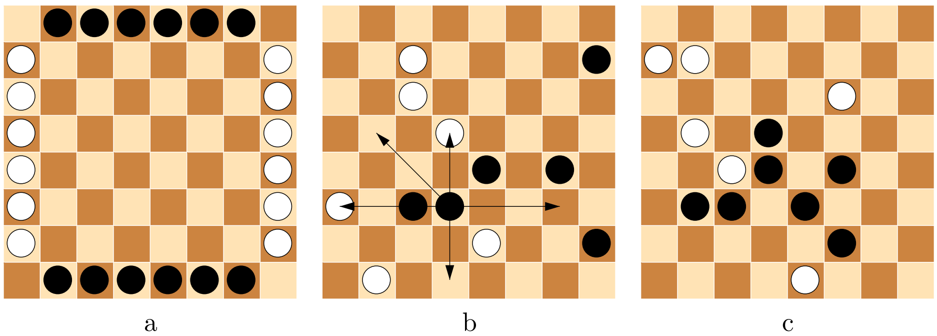

Figure 1: (a) The initial position of LOA. (b) An example of possible moves in a LOA game. (c) A terminal LOA position.

2.2 Evaluation function

The evaluation function used in MIA consists of seven features, whose weights will be tuned in section 4.1. Feature 1 is the *concentration*, which is computed 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 the piece is remote from the centre of mass. These distances are summed together, called

[1]MIA can be played at the website: http://www.cs.unimaas.nl/m.winands/loa/.

the sum-of-distances. Third, the sum-of-minimal-distances is calculated. It is defined as the sum of the minimal distances of the pieces from the centre of mass. This computation is necessary since otherwise boards with a few pieces would be preferred. For instance, if we have ten pieces, there will be always at least eight pieces at a distance of 1 from the centre of mass, and one piece at a distance of 2. In this case the total sum of distances is minimal 10. Thus, the sum-of-minimal-distances is subtracted from the sum-of-distances. Fourth, the average distance towards the centre of mass is calculated and the inverse of the average distance is defined as the concentration. Feature 2 is the *centralised centre-of-mass*. Positions with a somewhat more centralised centre of mass are preferred. Feature 3 is the *centralisation*. Pieces in the centre are preferred above pieces at the edges. Feature 4 is the *quad feature*. This feature looks at solid formations in the neighbourhood of the centre-of-mass by using quads. Details of this feature can be found in [16]. Feature 5 is the *mobility*. A bonus is given for the number of moves one has. Feature 6 is the *wall feature*. A wall is a group of pieces, which blocks the opponent's pieces at the edge. Position with walls are favoured. Feature 7 is the *side to move*.

2.3 Search Engine

MIA performs an $\alpha\beta$ depth-first iterative-deepening search. The program uses a *two-deep* transposition table [4], null moves [5] and PVS (Principal Variation Search) [9]. Before the null move is tried, the transposition table is used to prune a subtree or to narrow the window. For move ordering, the transposition move, if applicable, is always tried first. Next, two killer moves [1] are tried. All the other moves are ordered decreasingly according to their scores in the history table [13]. In the leaf nodes of the tree a quiescence search is performed. This quiescence search looks at capture moves, which form or destroy connections [16].

3 Learning methods

In this section we present the learning methods employed in MIA. These include temporal difference learning for tuning the weights of the evaluation function (subsection 3.1) and the Neural MoveMap heuristic to improve the move ordering (subsection 3.2).

3.1 Temporal difference learning

An attractive approach to learn an evaluation function is temporal difference (TD) learning (see [14]). Using this approach, each state s has associated a value V, representing the estimation of the expected outcome of the game. The state value can be used as an evaluation function in the search tree.

In the learning phase, the state values are updated so that they approach a target value. Let us consider a sequence of game positions $s_0, s_1, \ldots, s_T$. The target value for the final position, s_T, is given by

$$V^{target}(s_T) = \begin{cases} 1, & \text{if } s_T \text{ is a win for Black,} \\ 0, & \text{if } s_T \text{ is a draw} \\ -1, & \text{if } s_T \text{ is a win for White} \end{cases} \quad (1)$$

The target values for the non-terminal positions $s_0, s_1, \ldots, s_{T-1}$ are given by

$$V^{target}(s_t) = V(s_{t+1}) \quad (2)$$

To speed up TD learning, we can use TD(λ), which averages towards future target values:

$$V^{target}(s_t) = (1-\lambda) \sum_{k=1}^{T-t-1} \lambda^{k-1} V(s_{t+k}) + \lambda^{T-t-1} V(s_T) \quad (3)$$

λ taking values between 0 and 1.

In game programs using TD learning, V is typically represented by a parameterised function. To tune the weights of this function, we minimise the mean square of the TD error (i.e., $V^{target}(s_t) - V(s_t)$) with the following gradient updating rule:

$$\Delta w_t = \alpha(V(s_{t+1}) - V(s_t)) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial V(s_k)}{\partial w_i} \quad (4)$$

The gradient updating rule given above suggests a 'plain' back-propagation-like adaptation, but some of the improvements developed for supervised learning are likely to work for TD learning too.

To employ TD learning, we need to generate sequences of positions. Game sequences can be generated using game databases or games played by the learning program itself. In the latter case, a further choice can be made on the opponent. Possible options include using an already existing game program, playing against players with similar strength on the Internet, playing against itself, or using more learning players which improve their skill by playing together.

3.2 The Neural MoveMap heuristic

The Neural MoveMap (NMM) heuristic [6] is a recently developed learning method for move ordering. In the NMM heuristic a neural network is trained to estimate the likelihood of a move being the best in a certain position. During the search, the moves considered more likely to be the best are examined first. The essence of the heuristic is rather straightforward. However, the details of the heuristic are crucial for the heuristic to be effective, i.e., to be fast and to result in a small search tree. The details include: the architecture of the neural network, the construction of the training data, the training algorithm and the way the neural network is used for move ordering during the search.

A comparison of different architectures for the neural network is given in [6]. The authors found that the best architecture encodes the board position in the input units of the neural network and uses one output unit for each possible move of the game. When encoding a position we assign one input unit to each square of the board, with +1 for a black piece, −1 for a white piece and 0 for an empty square. An additional unit is used to specify the side to move. A move is identified by its origin and destination square (i.e., the current location and the new location of the piece to move). The activation value of an output unit corresponding to a move represents the score of that move. The resulting network has 65 input units and 4096 (64×64) output units. Although the network is very large, the move scores can be computed fast, since we have to propagate only the activation for the pieces actually on the board, and to compute only the scores for the legal moves. To increase the speed further, we do not use hidden layers. This way, the resulting move ordering requires just a little extra computation during the search, namely a summation over the pieces on the board.

A training instance consists of a board position, the legal moves in the position and the move which is the best. Of these three components, determining the legal moves by an algorithm poses no problem. The choices for the other two components are more difficult. In games where large game databases are available, an attractive choice is to use positions from these databases and to consider the one played in the game as the best move. In LOA, such databases are not available. The alternative is to generate positions by self-play, and to consider the one suggested by the game program as the best move.

The neural network described above performs a linear projection, and any training algorithm should thus be reasonably fast. Consequently, we can use any of the existent learning algorithms for neural networks without influencing significantly the training.

When the neural network is used during the search the moves are ordered according to the network's estimation of how likely a certain move is the best. The move ordering has to be placed in the context of the move orderings already existent in the game program. The solution employed in MIA is to replace in every node of the search tree the move ordering of the history heuristic by that of the neural network. A slightly better solution that combines the neural-network scores with history-heuristic scores is described in [7]. The solution employed in MIA is chosen for simplicity of implementation.

4 Experimental Results

In this section we test the improvement in MIA caused by the learning methods described in the previous section. Subsection 4.1 deals with tuning the evaluation function and subsection 4.2 with the move ordering.

4.1 Tuning the Evaluation Function

In subsection 2.2 we have seen that the evaluation function $E(s_t)$ of MIA is a parameterised function consisting of seven features. These features are already multiplied with weights to secure reasonable play. All the board positions occurring in a game are recorded and evaluated by $E(s_t)$. These raw values are converted to state values $V(s_t)$ by passing them to the hyperbolic tangent function [2]:

$$V(s_t) = \tanh(\beta E(s_t)) \tag{5}$$

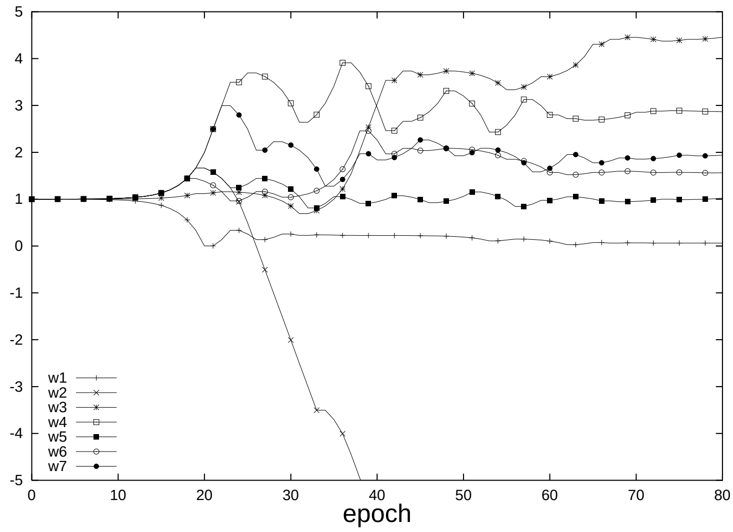

Figure 2: Development of the weights

where the constant β is chosen to ensure a not too steep evaluation function (i.e., $\beta = 0.0005$). In each position s_t the Δw_t is computed according to formula 4. Δw_t is accumulated over 1000 games (an epoch), after which the weights are updated (i.e., batch learning). As update rule for the weights we used RPROP [10]. In formula 4, λ is set to 0.8 because the evaluation function is already somewhat reliable (cf. [2]); α is replaced by the adaptive learning rates of RPROP.

To obtain results rather quick, the learning was performed by self-play using a four-ply deep search. One player changed its weights by TD learning, the other used the original weights. To avoid repeatedly the same play a small random factor was used in the evaluation function during the search. After each game the players switched side (to avoid overtuning). In figure 2 we plot the development of the weights during training. We see that the weights stabilise after approximately 60 epochs. The initial weight of the dominating centre-of-mass (*w1*) is decreasing to one tenth of its original value, indicating that this feature was overestimated. Interestingly, the weight for the centralised centre-of-mass feature (*w2*) is changing 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. If the centre-of-mass is in the centre, it is possible that pieces are scattered over the board (e.g., the initial position). 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. The weight of the centralisation component (*w3*) grows the most, indicating that this feature was underestimated.

After tuning the weights we tested the benefit of the new weights. A player with the new weights played 200 games against a player with the old weights, switching sides halfway. Each player had 60 seconds per move, simulating tournament conditions. In the second row of table 1 we give the match results. We observe that the modified version outplayed the original with a winning ratio of 1.68 (i.e., scoring 68% more winning points than the opponent).

	Score	Winning ratio
New Eval. vs. Old Eval.	125.5-74.5	1.68
NMM vs. History	102-98	1.04

Table 1: 200-game match results.

4.2 Optimising Move Ordering

In this subsection all experiments were performed with the original weights of the evaluation function. In order to use the NMM heuristic, we needed a training set. We used MIA to generate 600,000 position through self-play. During the games the program searched to a depth of four ply with a random component in the evaluation function. For each position the move played by the program was stored as the best move for that position. For training the neural network with the above generated training set we employed the RPROP algorithm [10]. The neural network obtained was included in MIA by replacing the history heuristic.

We compared the new move ordering, including the NMM heuristic, with the old move ordering that included the history heuristic. We used a set of 322 positions, which appeared in tournament play. In figure 3 we plot the relative performance of the two heuristics as the size of the search tree investigated using the new move ordering with that of the old one. We observe that the performance is decreasing until depth 5. After depth 6 the relative performance of NMM is improving with the depth. This pattern of the results was also noticed in [6]. At depth 11 (which is the regular search depth under tournament conditions) the reduction is 22 percent. The overhead of the NMM heuristic is 6 percent. Consequently, the effective time reduction is 17 percent.

A player using the NMM heuristic played 200 games under the same conditions as in subsection 4.1 against a player using the history heuristic. In the third row of table 1 we give the match results. The winning ratio of the NMM version was 1.04. Testing the combination of the NMM heuristic and the improved evaluation function will be part of future research.

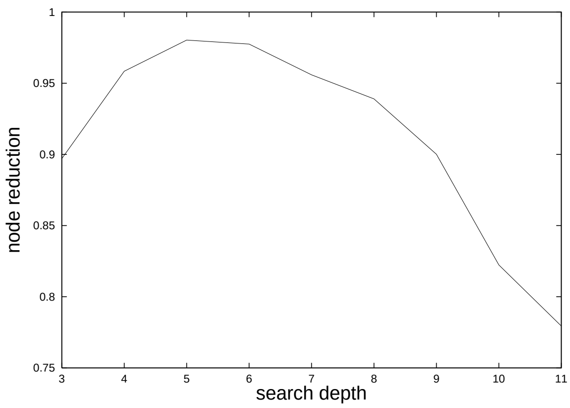

Figure 3: The performance of the NMM heuristic relative to the history heuristic.

5 Conclusions

This paper investigated the benefits of learning methods for the LOA tournament program MIA. We draw two conclusions based on our experimental results. First, TD learning is very beneficial for tuning the weights of the evaluation function. We found that three of our handcrafted components of the evaluation function were "wrong". By learning these were adjusted properly. Using the new evaluation function the program outperformed its previous version with a winning ratio of 1.68. Second, using the NMM heuristic the search of MIA was sped up with an effective time reduction of 17 percent, which led to a small improvement of play.

References

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

[2] J. Baxter, A. Tridgell, and L. Weaver. Experiments in parameter learning using temporal differences. *ICCA Journal*, 21(2):84–99, 1998.

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

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

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

[6] L. Kocsis, J.W.H.M. Uiterwijk, and H.J. van den Herik. Move ordering using neural networks. In L. Montosori, J. Váncza, and M. Ali, editors, *Engineering of Intelligent Systems, Lecture Notes in Artificial Intelligence, Vol. 2070*, pages 45–50. Springer-Verlag, Berlin, 2001.

[7] L. Kocsis, J.W.H.M. Uiterwijk, E.O. Postma and H.J. van den Herik. The Neural MoveMap heuristic in chess. *Proceedings of the Third International Conference on Computers and Games (CG'2002)*, 2002.

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

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

[10] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)*, pages 586–591, 1993.

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

[12] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

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

[14] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[15] G.J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

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