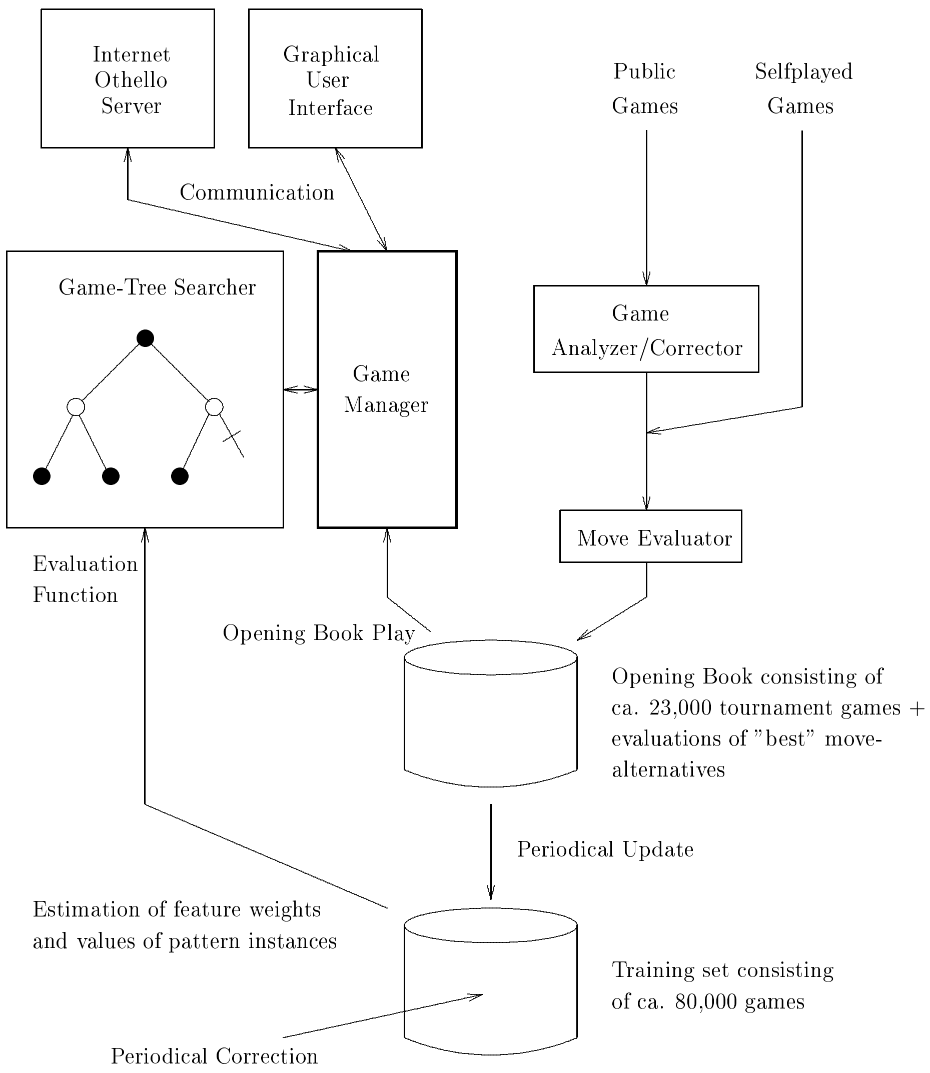
LOGISTELLO – A Strong Learning Othello Program

by Michael Buro, NEC Research Institute, Princeton, NJ

Overview

This note briefly describes LOGISTELLO — one of today's strongest Othello programs. Besides the powerful hardware on which it is running and the efficient implementation of standard game–tree searching techniques, the program's considerable playing strength is mainly due to several new approaches for the construction of evaluation features, their combination, selective search, and learning from previous games which the author has investigated in his Ph.D. thesis and considerably improved while working at NECI. Here is a graphical overview of LOGISTELLO's components:



LOGISTELLO's Evaluation Function

The classical approach for constructing evaluation functions for game–playing programs is to combine win correlated evaluation features of the position linearly:

$$f(p) = \sum_{i=1}^{n} w_i f_i(p).$$

This type of evaluation function is chosen very often since the combination overhead is relatively small compared to the time for computing the features and there are efficient methods available for determining the feature weights. When the relative importance of the features or even the feature set varies depending upon the game stage this simple model can be generalized to:

$$f(p) = \sum_{i=1}^{n_s} w_{s,i}\ f_{s,i}(p), \text{ where } s = \text{stage}(p).$$

Rosenbloom (1982) and Lee & Mahajan (1990) introduced a table–based evaluation scheme, in which values of all edge configurations were precomputed by (probabilistic) minimax algorithms and stored in a table for a quick evaluation of the edge structure. The pattern approach presented in [Buro 1997b] generalizes this technique by permitting the automatic evaluation of pattern configurations of any shape. The current pattern set is shown in Figure 1.

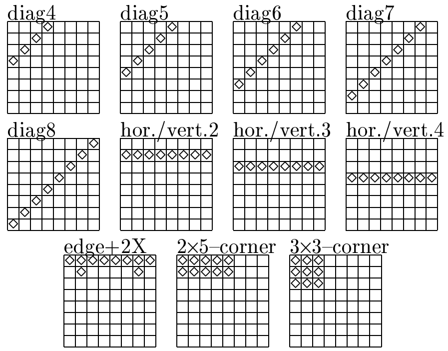


Figure 1: The current pattern set

LOGISTELLO's pattern features approximate important concepts in Othello, like striving for stable discs, maximizing the number of moves, and parity. The evaluation function is dependent upon game stage. In Othello the number of discs on the board is a reasonable measure. Thirteen stages were chosen, namely 13–16 discs, 17–20 discs, ..,61–64 discs. For pattern value estimation by means of linear regression a large set of training positions is used. It consists of ca. three million Othello positions stemming from about 60,000 games played between early versions of Igor Đurđanović's program KITTY and LOGISTELLO and 20,000 additional games that were generated by LOGISTELLO while extending its opening book. All positions were labeled with a disc differential by negamaxing the final game results in the tree built from all games. This procedure is accurate for endgame positions since the example games are played perfectly in this stage, whereas labels assigned to opening and middle–game positions are only approximations.

In addition to the pattern features shown in Figure 1 a simple phase dependent parity feature is used. Thus, LOGISTELLO's evaluation function has the following form:

$$\begin{aligned}
f(p) = (\, & [f_{\text{d4},s.1} + \dots + f_{\text{d4},s.4}] + [f_{\text{d5},s.1} + \dots + f_{\text{d5},s.4}] + \\
& [f_{\text{d6},s.1} + \dots + f_{\text{d6},s.4}] + [f_{\text{d7},s.1} + \dots + f_{\text{d7},s.4}] + \\
& [f_{\text{d8},s.1} + f_{\text{d8},s.2}] + [f_{\text{hv2},s.1} + \dots + f_{\text{hv2},s.4}] + \\
& [f_{\text{hv3},s.1} + \dots + f_{\text{hv3},s.4}] + [f_{\text{hv4},s.1} + \dots + f_{\text{hv4},s.4}] + \\
& [f_{\text{edge+2X},s.1} + \dots + f_{\text{edge+2X},s.4}] + \\
& [f_{2\times5,s.1} + \dots + f_{2\times5,s.8}] + \\
& [f_{3\times3,s.1} + \dots + f_{3\times3,s.4}] + f_{\text{parity},s})(p)
\end{aligned}$$

where $s = \text{stage}(p)$ and $f_{x,s,i}$ evaluates the i–th occurrence of pattern x on boards at game stage s (for instance, $f_{\text{edge+2X},s.1} + \dots + f_{\text{edge+2X},s.4}$ determines the evaluation for the entire edge structure by adding up table values for each of the four edges).

This entirely table–based evaluation function is very accurate and can be computed quickly.

PROBCUT: LOGISTELLO's Selective Search

Human players are able to find good moves without searching the game–tree in its full width. Using their experience they are able to prune unpromising variations in advance. The resulting game–trees are narrow and might be rather deep. By contrast the original minimax algorithm searches the entire game–tree up to a certain depth and even its efficient improvement — the $\alpha\beta$ algorithm — is only allowed to prune backwards because it has to compute the correct minimax value. The selective search procedure PROBCUT presented in [BURO 1995] permits pruning of subtrees that are unlikely to affect the minimax value and uses the time saved for analysis of crucial variations. The idea is to take advantage of the fact that values returned by minimax searches of different depths are highly correlated. In order to evaluate a position at height h, it can first be examined by a shallow search of depth $d < h$. The result v_d is then used for estimating the true value v_h and to decide with a prescribed likelihood whether v_h lies outside the current $\alpha\beta$ window (Figure 2). If so, the position is not searched more deeply and the appropriate window bound is returned. Otherwise, the deep search is performed yielding the true value. Here, a shallow search has been invested but relative to the deep search the effort involved is negligible.

A natural way to express v_h by means of v_d is to use a linear model of the form $v_h = a \cdot v_d + b + e$ with $a, b \in \mathbb{R}$ and a normally distributed error variable e with mean 0 and variance σ^2. After choosing height h and check depth d the parameters a, b and σ can be estimated using linear regression applied

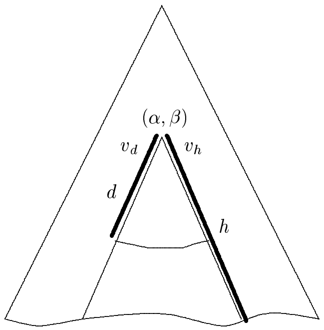


Figure 2: Forward cut scenario

to a large number of examples $(v_d(p_i), v_h(p_i))$. Now it is possible to test the cut conditions probabilistically: $v_d \geq \beta$ holds with probability at least p if and only if $(\hat{v}_h - \beta)/\sigma \geq \Phi^{-1}(p)$ is true, where $\hat{v}_h = a \cdot v_d + b$ and Φ denotes the distribution function of a normally distributed random variable with mean 0 and variance 1. This condition is equivalent to $v_d \geq (\Phi^{-1}(p) \cdot \sigma + \beta - b)/a$. Analogously, it can be shown that $v_h \leq \alpha$ holds with probability of at least p iff $v_d \leq (-\Phi^{-1}(p) \cdot \sigma + \alpha - b)/a$. If one of these conditions is met during the game–tree search the current position will not be searched to depth h. In this way large subtrees can be cut in order to save time for the relevant lines.

It remains to choose the cut threshold $\Phi^{-1}(p)$ suitably. For this purpose tournaments between the non–selective and the selective program version can be played using different thresholds in order to find the value that results in the greatest playing strength.

In the first PROBCUT implementation of LOGISTELLO $h = 8$ and $d = 4$ were chosen, and a, b, and σ were estimated separately for each game phase. When $\Phi^{-1}(p) = 1.5$ the winning percentage of the PROBCUT–enhanced version of LOGISTELLO playing against the brute–force version was 74.2% in a 70–game tournament.

Recently this selective search procedure has been improved. Experiments showed that by applying the following generalizations the playing strength can be increased even further:

1. allowing forward pruning at different heights. In this way bad moves — which exist in almost any position — or very good refutations can be detected earlier and more time can be saved for relevant lines.

2. performing several check searches of increasing depth until a cut condition is met. This procedure saves time in very unbalanced positions.

3. using different cut thresholds for each game stage.

LOGISTELLO's Opening Book Algorithm

If a player wants to be successful not only in a single game against an unknown opponent but in a sequence of games, he might be faced with simple but effective playing strategies of the opponent which cannot be met only by the well–known game–tree search techniques. Perhaps the most obvious and simple one is the following: "If you have won a game, try it the same way next time." A program with no learning mechanism and no random component follows this strategy, but is also a victim of it, since it does not deviate and therefore can lose games twice in the same way. In order to avoid this, it is necessary to find reasonable move alternatives. This can be accomplished passively, as the following strategy shows: "Copy the opponent's winning moves next time when colors are reversed." The idea behind this elegant method is to let the opponent show you your own faults in order to play the opponent's winning moves next time by yourself. In this way, even an otherwise stronger opponent can be compromised, since — roughly speaking — eventually he is playing against himself. Thus, copying moves makes it necessary to come up with good move alternatives actively. In order to do so, a player must have an understanding of his winning chances after deviations from known lines.

The mentioned basic requirements of a skilled match strategy lead directly to an algorithm for guiding opening book play based on negamax search. Suppose a game–tree is built from variations — starting with the initial game position — and its leaves are labelled as follows: The first component of the label indicates whether the corresponding position is a sure win, draw, or loss for the side to move (W,L,D). In cases where this classification is not yet known, a question–mark is used in the first component and the second component is the heuristic evaluation of the position computed, for instance by a deep negamax search (larger values indicate a higher winning chance for the side to move). Furthermore, in each interior node of the tree the heuristically best[a] deviation is added to the tree together with the reached position and its deep evaluation. Figure 3 shows examples. Here, solid lines mark variation moves, whereas long dashed lines represent the best move alternatives in each interior node. Short dashed lines indicate the existence of other deviations with lower evaluation which don't have to be considered for the moment because the best deviation would be preferred.

Given such a tree, it is easy to guide the opening book play in best–first manner: Find the node

[a]In what follows, *best* means *heuristically best.*

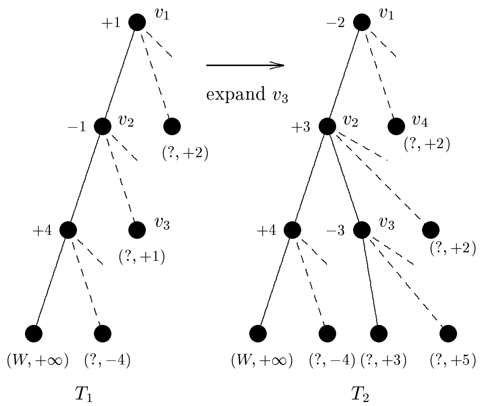


Figure 3: Example opening book trees. The principal leaf v_3 of T_1 is chosen for expansion. In the resulting tree T_2 leaf v_4 would be expanded next.

corresponding to the current position, propagate the heuristic evaluations from the leaves to that node by means of the negamax algorithm, and choose the move that leads to the successor position with lowest evaluation.[a] Before the move to be played can be determined using this algorithm, heuristic evaluations have to be assigned to the leaves for which the outcome is known. A natural choice is $+\infty$ for won positions, 0 for draws, and $-\infty$ in lost positions. Provided that heuristic evaluations are always greater than $-\infty$, and there are still unexplored variations, this setting ensures that games will not be lost twice in the same way.

The algorithm applied to the root of the example–tree T_1 in Figure 3 yields the optimal path (v_1, v_2, v_3) which maximizes the winning chances of the root player against best counter-play (local to the tree).

Knowing how to select moves from an opening book immediately enables the automatic extension of the book by iterated expansion of the leaf at the end of the current principal variation. In each step, the best move and the best deviation are determined in the leaf position and added to the tree together with their evaluations. Furthermore, if the leaf position was reached by a deviation, the next best deviation in the predecessor position also has to be found and added. Thus, after each expansion, the tree is complete again consisting of a variation skeleton augmented by the best deviation in each interior node. Figure 3 illustrates this process.

[a]Since in this process heuristic evaluations of positions from different game phases are compared, it is recommended to use an evaluation function with a game–phase independent meaning, such as winning probability or expected result.

Milestones

1991 first experiments with statistical feature combination techniques
1992 searching for significant and fast evaluation features for Othello
1993 LOGISTELLO wins its first tournament (Paderborn I)
1994 - ProbCut - a new selective search approach
- opening book learning
- LOGISTELLO wins Paderborn II

1995 - 10-disc patterns
- improved opening book algorithm
- LOGISTELLO wins Paderborn III

1996 - multi-ProbCut
- hash-table improvement
- LOGISTELLO wins Paderborn IV

1997 - construction of a much better evaluation function
- LOGISTELLO plays a 6-game match against the current World-Champion Takeshi Murakami

Recent Improvements

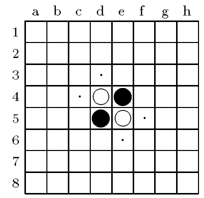
3/1995	10-disc patterns
5/1995	new best-first style endgame search
7/1995	book–algorithm now maximizes the out-of-book score
5/1996	book–randomization
10/1996	multi–ProbCut
11/1996	major table estimation bug fixed, new version beats old 57% of the time
12/1996	hash-table improvement saves 15-40% searchtime
3/1997	a new table estimation technique increases the playing strength considerably
4/1997	the entire book has been re-computed using the new evaluation function
7/1997	increasing the number of training examples and generating rare positions to fill table gaps has improved the evaluation function even further

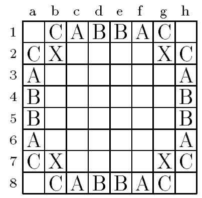
LOGISTELLO Summary

Eval. Features:	- game stage dependent pattern tables - a simple parity measure
Feature Comb.:	- linear
Search:	- NegaScout - corner quiescence search - multi–ProbCut - iterative deepening
Move sorting:	- hash-tables containing moves and value bounds - response killer lists - shallow searches
Search speed:	- midgame: ca. 160,000 nodes/sec - endgame: ca. 480,000 nodes/sec
Search depth:	in a 2x30 minutes game: - midgame: 18–23 ply (selective) including 10–15 brute-force ply - endgame: win/loss/draw at 26–22 empty squares, exact score 1-2 ply later
Opening Book:	- consists of ca. 23,000 games + evaluations of "best" move alternatives - is automatically updated
Miscellany:	- thinking on opponent's time - communication with IOS via socket or with graphical interface via file system
Language:	C
OS/compiler:	Linux/gcc 2.7.2.1
Machine:	PentiumPro/200 MHz

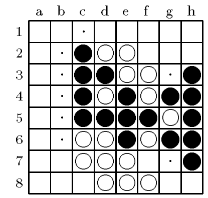
Othello

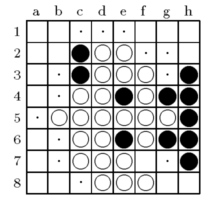
The rules of Othello are simple: It is a two–person zero sum perfect information game played on a 8×8 board using 64 two–colored discs. Black goes first with Black and White alternating moves thereafter — if possible. In order to move, a disc is placed on an empty square showing the player's color such that the new disc and another disc already on the board bracket at least one opponent's disc. All bracketed discs in all directions must be flipped now showing the player's color. An example is shown in the diagrams to the right. A player with no legal moves must pass. The game ends if neither player has a legal move. The player who has the disc majority wins the game.


Start position



Square notation


White to move


Position after move b5

References

[Buro 1995] *ProbCut: An Effective Selective Extension of the Alpha–Beta Algorithm*, ICCA Journal 18(2), 71–76.

[Buro 1997a] *Toward Opening Book Learning*, NECI Technical Note #2.

[Buro 1997b] *Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello*, NECI Technical Report #96.

[Lee & Mahajan 1990] *The Development of a World Class Othello Program*, Artificial Intelligence 43, 21–36.

[Rosenbloom 1982] *A World–Championship–Level Othello Program*, Artificial Intelligence 19: 279–320.