THE EVOLUTION OF STRONG OTHELLO PROGRAMS

Michael Buro
NEC Research Institute
4 Independence Way, Princeton NJ 08540, U.S.A.
mic@research.nj.nec.com

Abstract This paper surveys the evaluation and search techniques utilized by the strongest Othello programs of their time during the past twenty years. In this time span computer Othello has experienced considerable progress which culminated in the convincing 6-0 victory of LOGISTELLO against the then World-champion Takeshi Murakami in 1997. The focus of this article is the evolution of Othello evaluation functions and heuristic search techniques which quite nicely reflect the general A.I. trend of replacing slow and error-prone manual tuning by automated machine learning approaches.

Keywords: Computer Othello, Evaluation Function Learning, Selective Search

1. Introduction

Today, machines play the Japanese board game Othello (Fig. 1) very well. In fact, the 6-0 defeat of the then human World-champion Takeshi Murakami by LOGISTELLO in 1997 (Buro, 1997) strongly indicates that machines have surpassed human playing strength. The first computer Othello tournaments took place in 1979, about eight years after Goro Hasegawa "invented" Othello by slightly changing the rules of Reversi – a much older game from England. In 1980 the first man-machine Othello tournament was organized by Peter Frey at Northwestern University. It was won by World-champion Hiroshi Inouie, who – however – lost a game against "The Moor" by Mike Reeve and David Levy. This marks the first time a World-champion lost a game of skill against a computer. In the 1980s Othello programming became very popular in France, the U.K., The Netherlands, and the U.S. Programs were getting stronger and began to dominate man-machine events as early as 1989 when five programs defeated five human players – including three of the four best players of the time – by 12:8 in London. The two man-machine events

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35660-0_65
R. Nakatsu et al. (eds.), *Entertainment Computing*

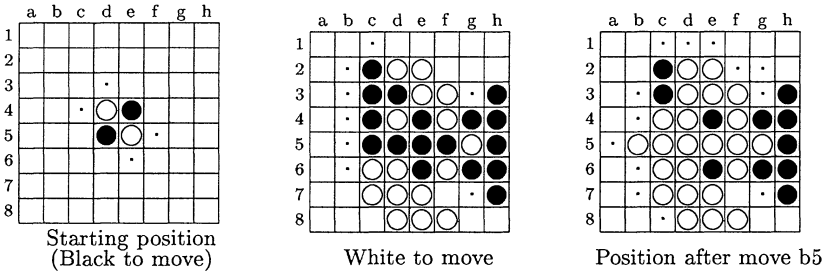

Figure 1. Othello is a popular Japanese board game played by two players on an 8x8-board using 64 two-colored discs. Moves consist of placing one disc on an empty square (e.g. white on b5 on the second board) and flipping all – and at least one – bracketed opponent's discs over (c5..f5,c4,d2). When neither player has a legal move the game ends and the player with the disc majority wins.

that followed 1992 and 1994 in Paris indicated that machine Othello had grown out of its infancy: the machines clearly won by 29.5:19.5 and 32:4, respectively. In 1997, it became clear that machine Othello had surpassed human abilities: LOGISTELLO defeated the reigning World-champion Takeshi Murakami 6:0 in a match of six long-timed games.

As representatives for the many good Othello programs with unique features that have been written in the past 25 years we have selected three programs that dominated computer Othello for a while. We survey the advances in evaluation function construction, heuristic search, and opening book learning. The programs are: IAGO (Rosenbloom, 1982), BILL (Lee and Mahajan, 1990), and LOGISTELLO (Buro, 1999a, 1999b, 2000).

2. Evaluation Function Evolution

In mini-max search evaluation functions are used to estimate the winning chance for the player to move. One way of constructing such functions is to select distinct numerical properties of the game state – so called features – which are correlated with winning, and combine them to obtain a single numerical value. Selecting features is one of the most important and difficult sub-tasks in the construction of game-playing programs. It requires both domain-specific knowledge and programming skills because of the well-known tradeoff between speed and knowledge in look-ahead search.

The most important positional features in Othello are disc **stability**, **mobility**, and **parity**. **Stable** discs can not be flipped by the opponent and contribute directly to the final score. The most prominent stable discs are occupied corners, which can be used as anchors for creating more stable discs. Having fewer **move options** than the opponent is dangerous because one may run out of good moves sooner. Finally, making the last move in an Othello game is advantageous since it increases one's own disc count while decreasing the number of opponent's discs. **Parity** generalizes this observation by considering last move opportunities for every empty board region.

Iago (1982): a classic, hand-crafted evaluation function

IAGO's evaluation function has the following form:

$$\begin{aligned} e(p) &= C_1(p) \cdot \text{EdgeStability}(p) + 36 \cdot \text{InternalStability}(p) + \\ &\quad C_2(p) \cdot \text{CurrentMobility}(p) + 99 \cdot \text{PotentialMobility}(p), \end{aligned}$$

where the application coefficients $C_1(p)$ and $C_2(p)$ are piece-wise linear and non-decreasing in MoveNumber(p). **EdgeStability** is a table based evaluation pre-computed by an iterative algorithm. It assigns heuristic values to filled edges and fills the remaining table entries by applying a formula which takes values of successors and square access probabilities into account. **InternalStability** is computed by an iterative algorithm which proves disc stability beginning with corners. **CurrentMobility** measures the relative mobility by counting legal moves for both sides (a, b) and combining the values by a rational function: truncate($1000 \cdot (a-b)/(a+b+2)$) (1). The **PotentialMobility** term combines three future mobility measures: the number of frontier discs, the number of empty squares adjacent to opponent's discs, and the sum of the number of empty squares adjacent to each of the opponent's discs. Each measure computes two values which are combined similar to equation (1). The three values are then summed to yield a single value.

IAGO's features have been chosen based on a careful analysis of Othello, while most evaluation parameters have been guessed.

Bill (1990): partly pattern based, feature weights are learned

BILL's evaluation is based on the following four features: EdgeStability, CurrentMobility, PotentialMobility, and SequencePenalty. Similar to IAGO a probabilistic mini-max algorithm is used to pre-compute **edge stability** values which are stored in a table. **CurrentMobility** is calculated by summing up move-square weights and adjusting the sum by penalty terms depending on whether the moves capture/surrender a corner, how many discs and in how many directions they are flipped, and whether flipped discs are internal or on the frontier. **PotentialMobility** measures the likelihood of future move options by examining the adjacency between empty squares and discs. The **SequencePenalty** feature recognizes long disc sequences of equal color along lines and assigns a negative value to them depending on the board location. The evaluation speed of all four features is considerably increased by an extensive use of pre-computed tables which are indexed by horizontal, vertical, and diagonal lines of the board.

To form a single value evaluation the four features are combined using a quadratic discriminant function. Assuming a multivariate normal distribution of the features, the intra-class feature density can be expressed as follows:

$$p(\boldsymbol{x} \mid C) = (2\pi)^{-n/2} |\boldsymbol{\Sigma}_C|^{-1/2} \exp\Big\{ -\tfrac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_C) \boldsymbol{\Sigma}_C^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_C)' \Big\}$$

with n-dimensional feature vector $\boldsymbol{x}$, mean feature vector $\boldsymbol{\mu}_C$ and feature covariance matrix $\boldsymbol{\Sigma}_C$ for the position class $C \in \{W, L\}$ (win/loss). From Bayes' rule it follows $P(W|\boldsymbol{x}) = 1/(1+\exp(-e(\boldsymbol{x}))$, where e($\boldsymbol{x}$) is the following quadratic discriminant function which is used to evaluate Othello positions in BILL:

$$e(\boldsymbol{x}) = (\boldsymbol{x}-\boldsymbol{\mu}_W)\boldsymbol{\Sigma}_W^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_W)'-(\boldsymbol{x}-\boldsymbol{\mu}_L)\boldsymbol{\Sigma}_L^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_L)'+\log|\boldsymbol{\Sigma}_W|-\log|\boldsymbol{\Sigma}_L|$$

Covariance matrices and mean vectors for 60 game stages are estimated from sample positions to end up finally with an evaluation function vastly superior to IAGO's: BILL wins 100% of its games versus IAGO with only 20% as much thinking time.

Logistello-1 (1994): pattern values learned independently

BILL's evaluation showed a strong performance but still relies on many manually-tuned (or guessed) parameters. Moreover, the evaluation of quadratic forms is time-consuming if the features themselves can be evaluated very quickly (e.g., by table look-up). Finally, features are not necessarily multivariate normally distributed – for instance in the presence of binary features. The main ideas behind LOGISTELLO's first evaluation are to learn pattern values from sample data – rather than guessing them – and to drop the normal distribution requirement by using logistic regression to combine the following features (Buro, 1995b): CurrentMobility & PotentialMobility (both approximated by line patterns) and estimated pattern values for all horizontal, vertical, diagonals of length 5..8, and the 2x4-corner region (Fig. 2). Pattern values are estimated independently by looking at all sample positions that match the particular pattern and averaging all position scores – which are assigned by propagating the final disc differential backward to all intermediate game positions. Pattern tables are estimated for 13 game stages dependent on the number of discs on the board. Finally, game-stage-dependent weights are assigned to all features by logistic regression. The evaluation function form is quite similar to that obtained by discriminant analysis: $P(W \mid \boldsymbol{x}) = 1/(1+\exp(-\boldsymbol{x}\boldsymbol{\beta}))$. However, logistic regression does not require the features to be multivariate normal – even the use of discrete

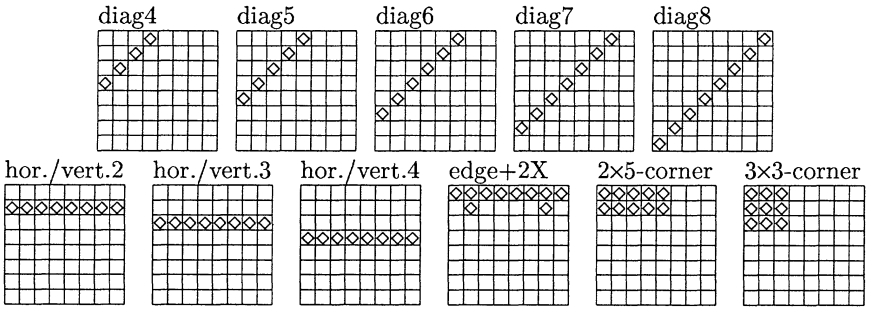

Figure 2. LOGISTELLO's current pattern set.

features with a small range is possible – and the resulting evaluation function is linear. A small drawback is the necessity to solve a system of nonlinear equations for finding the maximum-likelihood weight estimates. But this is a one-time task which can be performed off-line.

The resulting evaluation function is entirely table based and therefore an order of magnitude faster than those similar to BILL's and IAGO's. Furthermore, *all* evaluation parameters are learned from sample positions which frees program authors from guessing parameters or laborious manual tuning. Using this kind of evaluation function, LOGISTELLO dominated the computer Othello scene until 1996 when HANNIBAL entered the scene.

Logistello-2 (1997): joint learning of pattern values

With hindsight the next evolutionary step is obvious: the 1994 approach of assigning pattern values by independent estimations neglects pattern correlations which are crucial. For instance, the value of a corner disc is overestimated because corner squares are covered by multiple patterns. The solution is to let a large sparse linear regression assign values to pattern configurations by fitting the position labels and treating pattern configurations as binary variables. Linear regression takes care of feature correlations and at the same time can assign arbitrary values to patterns whose meaning is not bound by limited human understanding of the problem (e.g., "Mobility" and "PotentialMobility").

LOGISTELLO's current evaluation function still distinguishes 13 game stages, depending on the number of discs on the board. It has the following form:

$$\begin{aligned} e(p) = ([&e_{\mathrm{d4},s.1} + \ldots\ e_{\mathrm{d4},s.4}] + [e_{\mathrm{d5},s.1} + \ldots\ e_{\mathrm{d5},s.4}] + [e_{\mathrm{d6},s.1} + \ldots\ e_{\mathrm{d6},s.4}] \\ +&[e_{\mathrm{d7},s.1} + \ldots\ e_{\mathrm{d7},s.4}] + [e_{\mathrm{d8},s.1} + e_{\mathrm{d8},s.2}] + [e_{\mathrm{hv2},s.1} + \ldots\ e_{\mathrm{hv2},s.4}] \\ +&[e_{\mathrm{hv3},s.1} + \ldots\ e_{\mathrm{hv3},s.4}] + [e_{\mathrm{hv4},s.1} + \ldots\ e_{\mathrm{hv4},s.4}] + [e_{\mathrm{e+2X},s.1} + \ldots\ e_{\mathrm{e+2X},s.4}] \\ +&[e_{2\times5,s.1} + \ldots\ e_{2\times5,s.8}] + [e_{3\times3,s.1} + \ldots\ e_{3\times3,s.4}] + e_{\mathrm{parity},s})(p), \end{aligned}$$

where $s = \mathrm{game\_stage}(p)$ and $e_{x,s.i}$ evaluates the i-th occurrence of pattern x on boards at game stage s (e.g., $e_{3\mathrm{x}3,s.1} + .. + e_{3\mathrm{x}3,s.4}$ determines the evaluation for the whole corner structure by adding up table values for each of the four corners). In addition to the patterns shown in Figure 2 a simple parity (pattern) feature is used which deals with the last move advantage globally by considering the number of empty squares modulo 2. Several million training positions labeled with either the true mini-max value or an approximation were generated from self-played games to fit approximately 1.2 million weights. This figure takes weight sharing among symmetric configurations into account. Experiments in form of tournaments have shown that the strength increase compared to the 1994 evaluation function is equivalent to a speed-up factor of about ten, which is otherwise only achievable by parallelization.

In this article many details and possible generalizations of the presented pattern learning approach are not dealt with. The interested

reader is referred to (Buro, 1999a) which introduces a generalized linear evaluation model (GLEM) and discusses theoretical and implementation issues of pattern learning. GLEM combines automatic feature space exploration with fast numerical parameter tuning by building patterns from atomic features and assigning pattern weights by linear regression.

3. Heuristic Search

IAGO and BILL use classic techniques for improving the performance of the plain α-β algorithm: both increase the search depth iteratively to make use of move-ordering information gathered in previous iterations. IAGO keeps the first three plies in memory to have access to the previously determined best moves and uses response killer lists for move ordering in deeper nodes. BILL utilizes hash and killer tables for this purpose. In addition, it performs zero-window searches which slightly decrease the number of searched nodes compared with the usual α-β algorithm. Neither program performs quiescence searches or any other form of search extensions. The 1994 version of LOGISTELLO employs the same techniques as BILL but also uses shallow searches for move sorting and makes extensive use of a selective search heuristic called PROBCUT (Buro, 1995a). PROBCUT permits pruning of subtrees that are unlikely to affect the mini-max value and uses the time saved for analysis of probably more relevant variations. This approach capitalizes on the fact that values returned by mini-max searches of different depths on the same subtree are highly correlated, provided that a reasonably good evaluation function and, if necessary, a quiescence search is used. In this case, a shallow search result v_s is a good predictor for the deep mini-max value v_d. Based on this estimation, it can be determined whether the deep mini-max value lies outside the current α-β window with a prescribed probability. If so, the position need not be searched more deeply because the deep search result will unlikely change the root's mini-max value. 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, due to the exponential tree growth. Using s=4 and d=8 the PROBCUT version scores 74% of the points in a 70-game tournament against full-width search.

Although PROBCUT already marks a large improvement over full-width α-β search, it can easily be refined in several ways: MULTI-PROBCUT (MPC, Buro, 2000) allows for pruning at different search depths, uses game-stage dependent cut thresholds, and performs shallow check searches using iterative deepening. The latter improvement detects extreme positions much earlier. Incorporated in the 1997 version of LOGISTELLO, MPC featuring up to ($s=5$, $d=17$) cuts and two cut thresholds (for the opening and middle game) beats regular ($s=4$, $d=8$) PROBCUT by about 72% in a 140 game tournament and ties with the

full-width search version when its time gets reduced by a factor of 25. All of today's strong Othello programs employ MPC variants.

4. Other Important Improvements

In this section we sketch two other areas of progress: opening book learning and selective endgame search. Opening books are collections of move sequences that guide game programs through the opening phase. They serve three purposes: 1) time can be saved by instantly playing moves stored in the opening book, 2) following sound opening lines avoids falling into known strategic traps, and 3) opening book learning guards against losing a game twice in the same way. Opening book learning was pioneered in LOGISTELLO (Buro, 1999b) and is now used by all good Othello programs.

Endgame search is very important in Othello because it enables machines to play optimally when a small number of moves remains to be played in the game and look-ahead search therefore can reach terminal positions. Depending on the current position and the remaining time programs first switch from middle-game search to outcome search trying to determine the winner of the game. Afterwards, the score is maximized by an exact search. The purpose of selective endgame search is to find winning or drawing moves as quickly as possible even before determining the winner. LOGISTELLO's selective endgame search is based on the PROBCUT idea.

5. Summary and Outlook

This article has shown how Othello programs evolved from classic hand-tuned to sophisticated learning systems which have surpassed human playing strength. Automatic tuning millions of evaluation and selective search parameters and autonomously expanding opening books considerably increases the expressiveness of evaluation functions and frees programmers from laborious manual tuning. Looking at the success of modern Othello programs it seems that programs for other games could also benefit from the applied techniques. For instance, it is known that chess and go players make use of extensive pattern knowledge and perform sophisticated selective searches. A straight forward application of the presented pattern learning technique is possible for games for which pattern evaluations can be constructed quite naturally from the raw board representation – such as 8x8 Othello, 8x8 Lines-Of-Action, and backgammon. For games like chess and go an additional feature layer is necessary for covering local and dynamic position properties: positions are mapped into an intermediate feature set which is then used to form patterns. In the work on Othello evaluation functions the problem of finding these so-called atomic features has been largely ignored because patterns formed by small sets of squares already capture important Othello features quite well. Constructing atomic features from the

game definition is a challenging research problem. However, it appears to be simpler than generating general features because the GLEM system can automatically construct short feature combinations and tune a large number of parameters automatically. One promising approach to finding atomic features is genetic programming. Future research will show if this idea in conjunction with GLEM can lead to improved evaluation functions not only for games.

Othello has been a fruitful test-bed for A.I. research. Although nowadays the game is dominated by strong programs, there is no sign of diminishing interest by human players. On the contrary – the number of players participating in the annual (human) World-championships is growing and the level of play is rising. This is certainly a consequence of freely available Othello software and internet Othello servers, which help players to refine openings, middle-game strategy, and even endgame tactics. After a long period of inactivity, recently the computer Othello community has begun to show life signs again. Several programmers with new ideas – ranging from vastly improved endgame speed to alternative ways of estimating pattern values – have connected new programs to GGS to compete with the old-timers like LOGISTELLO and KITTY (written by Igor Durdanovic) which have not been worked on for years (but still do quite well). There is also a growing interest in new game formats – such as the synchro-rand mode in which two games are played simultaneously starting with the same random starting position and reversed colors – and larger board sizes like 10x10. Larger boards create new challenges – not only for humans – as the described pattern learning technique requires refinement due to the need for larger patterns.

References

Buro, M. (1995a). ProbCut: An effective selective extension of the alpha-beta algorithm. *ICCA Journal*, 18(2):71–76.

Buro, M. (1995b). Statistical feature combination for the evaluation of game positions. *JAIR*, 3:373–382.

Buro, M. (1997). The Othello match of the year: Takeshi Murakami vs. Logistello. *ICCA Journal*, 20(3):189–193.

Buro, M. (1999a). From simple features to sophisticated evaluation functions. In van den Herik, H. J. and Iida, H., editors, *Computers and Games, Proceedings of CG98, LNCS 1558*, pages 126–145. Springer-Verlag.

Buro, M. (1999b). Toward opening book learning. *ICCA Journal*, 22(2):98–102.

Buro, M. (2000). Experiments with Multi-ProbCut and a new high-quality evaluation function for Othello. In van den Herik, H. J. and Iida, H., editors, *Games in AI Research, Proceedings of a workshop on game-tree search held in 1997 at NECI in Princeton, NJ*, pages 77–96. Universiteit Maastricht, The Netherlands.

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

Rosenbloom, P. (1982). A World–championship–level Othello program. *Artificial Intelligence*, 19:279–320.

3. HOME/ARCADE GAMES AND INTERACTIVE MOVIES