Innovative Opening-Book Handling

Chrilly Donninger[1] and Ulf Lorenz[1,2]

[1] HydraChess.com
{chrilly, ulf}@hydrachess.com
[2] Department of Computer Science,
Universität Paderborn, Germany
flulo@uni-paderborn.de

Abstract. The best chess programs have reached the level of top players in the human chess world. The so-called opening books, which are databases containing thousands of Grandmaster games, have been seen as a big advantage of programs over humans, because the computers do never forget a variation. Interestingly, it is this opening phase which causes most problems for the computers. Not because they do not understand openings in general, but because the opening books contain too much rubbish. We introduce a heuristic which explores the database during a game. Without that, the computer repeats failures of weaker players. Our contribution presents best practice.

1 Introduction

Game playing is one of the core topics in Artificial Intelligence (AI). Unequivocal success stories come along with this topic. In some of the most popular parlor games, the computer players have already conquered the crown, such as in Checkers [19] or Othello [3]. Some games like Connect-4 have even been completely solved [21] which means that there exists an algorithm that for all positions of the game is able to play the perfect move. The game of Go is still out of the machine's reach, but in computer chess we nowadays see the leading programs crossing the borderline to the human top players. For instance, the current human vs. machine team-chess world-championship is in the hands of the machines [13]. Although our new heuristic is certainly powerful in any 2-person zero-sum parlor game, we restrict our description for the sake of clarity to the example of Chess, the most famous board game.

1.1 How a Chess Programs Works

The key feature, which enables current chess programs to play as strong as, or even stronger than the best human beings, is their search algorithm. The programs perform a forecast. Given a certain position, the line of reasoning is: what can I do, what can my opponent do next, what can I do thereafter? Modern programs use some variants of the so-called Alphabeta algorithm [12] in order to examine the resulting game tree. This algorithm is efficient in the sense that in most cases it will examine only $O(b^{d/2})$ leaves instead of b^t leaves, assuming a game-tree depth d and a uniform branching factor b. With the help of upper and lower bounds, the algorithm uses information that

H.J. van den Herik et al. (Eds.): ACG11, LNCS 4250, pp. 1 – 10, 2006.

it collects during the search process in order to keep the remaining search tree small. This results in a sequential procedure which is difficult to parallelize, and naive approaches waste resources.

Although the Alphabeta algorithm is efficient, we are not able to compute true values for all positions in games like Chess, because the game tree is far too large. Therefore, computers apply tree search as an approximation procedure. This works as follows. First of all, a partial tree rooted near the top of the complete game tree is selected for examination. Usually, this selection is based on a maximum depth parameter. Then they assign heuristic values (e.g., one side having a Queen more means that this side will probably win) to the artificial leaves of the pre-selected partial tree and propagate these values up to the root of the tree as if they were true ones. When Shannon [20] proposed a first architecture for a computer chess program – which is in its core the same architecture as still used today [14] – it seemed quite natural that deeper searching leads to better results. This, however, is by far not self-evident, as theoretical analyses by Nau [16], Althöfer [1], or Beal [2] show. Nevertheless, the key observation over the last 40 years in most games like Chess is that a game tree acts as an *error filter*. The larger the tree which we can examine, and the more sophisticated its shape, the better is its error-filter property. New interesting ideas like in [11] have not reached practice yet.

1.2 The Race Between the Top-Four

Let us briefly summarize the breathtaking modern computer-chess history over the last 35 years. Before, between 1940 and 1970, programmers attempted to mimic human chess style, but the resulting programs were weak. Then in the 1970s, CHESS 4.5 was the first 'strong' program, emphasizing tree search. It was the first one that could win a tournament against humans, in the Minnesota Open 1977. In 1983, the chess-machine BELLE became National Master, with a rating of 2100 Elo[1]. In 1988, HITECH won for a first time against a Grandmaster, and in the same year DEEP THOUGHT already played on Grandmaster level itself. The world changed in 1992 when the CHESSMACHINE, a conventional PC program by Ed Schröder became World Champion. IBM's DEEP BLUE [10] beat Kasparov in a 6-game match, five years later [18].

Interestingly, since 1992 only PC programs have been World Champions. They have dominated the world, increasing their playing strength by about 30 Elo points per year. Nowadays, the computer-chess community is highly developed. Everything desirable exists, even a Virtual Reality with special machine rooms, closed and open tournament rooms etc. Anybody can play against Grandmasters or strong machines via Internet. At the moment, programs are at the point that they supersede their human counterparts. Four chess programs have a race for the chess crown. (1) SHREDDER**,** by Stefan Meyer-Kahlen, has been the dominating program over the last decade. (2) FRITZ**,** by Frans Morsch is the best-known program, (3) JUNIOR**,** by Amir Ban and Shay Bushinsky, is the current Computer-Chess World Champion, and last but not least (4) HYDRA [6], which is certainly the strongest program at the moment. These

[1] Elo: statistical measure. 100 Elo points difference correspond to 64% winning chance. Beginner ~1000 Elo, Int. Master ~2400, Grandmaster ~2500, World Champion ~2830.

four programs (indeed not with HYDRA, but some predecessor) scored more than 95% of the points against their opponents on the World Championship 2003.

1.3 Organization of This Paper

Besides search the existence of opening books is important in computer chess. We start emphasizing the importance. Then we discuss the techniques, which are traditionally used to generate opening databases. Thereafter, we present our own ideas and, last but not least, we show the effectiveness of the new heuristics. Our main idea is that the number of how often a move has been played in a certain position gives us a good hint of how risky it is to play this move; and that the success, with which a move has been played in the past, gives us a good hint on its potential. How these hints can be related to a specific position should be computed by the best available chess expert: the chess program.

2 Opening Books

The opening books of chess programs are large databases which contain thousands or millions of early positions together with one or several proposed moves for each of the positions. It is called 'opening book', because it contains early positions of the game, all taken from previously played games. The databases are partially generated with the help of known grandmaster games, and partially consist of analyses of chess experts. They are of great importance because (1) they can be computed off-line and (2) the top Grandmasters make so very few mistakes that they can lead a clear opening advantage to victories (thus a program should prohibit that a Grandmaster has a clean advantage after the opening phase).

2.1 The Situation

The opening books have always been an important feature in chess programs. For instance, in 1995, the program FRITZ beat IBM's DEEP BLUE, and later won the World Championship, because DEEP BLUE's opening book ended earlier than FRITZ's, and DEEP BLUE played a short castling directly after the book was finished. This mistake was so bad in a long term sense that Fritz had no difficulties to win the game.

Within the last decade, many people tried to generate top-level books for chess programs, but only three of them have been successful and have become famous in the computer chess community.

Over the years, these book writers increased their importance any further. Most of the games between top programs were decided in the opening. In the World Championship in Graz, 2003, the situation became nearly absurd. When FRITZ played against SHREDDER, SHREDDER left the book with a difficult position. Subsequently, FRITZ won. Indeed, the pairing seemed not to be FRITZ vs. SHREDDER, but Mr. Necci vs. Mr. Kure (the two most successful book writers) in a remote fight. Later, SHREDDER played against BRUTUS and SHREDDER won because of the opening book. A strange detail was that Kure made the opening book both for FRITZ and for BRUTUS. In the game FRITZ vs. BRUTUS, the programmers of both programs had in use the same opening book by Kure, but FRITZ' programmer made the mistake to add a change.

This was the decisive change which brought BRUTUS an advantage. Subsequently, BRUTUS won. In another game, JUNIOR played against SHREDDER, and SHREDDER stayed in the opening database until the final draw! As a direct consequence, some people predicted the death of computer chess. They assumed the influence of the book writers so strong that they believed that the books would become the dominating factor when two programs would play each other.

In order to find out what the book writers really do we interviewed most of the involved people. They are definitively not the strongest chess players in the world, but range between 2200 and 2550 Elo themselves.

2.2 State of the Art

From the interviews we learned that the top book writers used a mixture of different heuristics in which the following components seemed to play an important role.

(1) **Personal intuition.** This mainly seems to be used (a) to weight the points (2), (3), and (4) and (b) to decide on the 'main systems' which the program should play. (In Chess, the opening phase is divided into so-called 'systems' which in the course of time have names such as 'Sicilian Defence' and others.)

(2) **Their own chess expertise.** The writers seem to be convinced that they are able to improve the decisions of any chess program in some positions, of which they say that the machine does 'not understand' them.

(3) **Test-games.** All computers play in a similar manner. So, if you find a way to beat one program, you will have good chances that another program will lose in the same way.

(4) **Statistical information.** For each position played in the past, the following information is available, (a) which moves have been played in the past, (b) how strong the players who played the move are, and (c) how the games after the moves ended. For example: after 1. e4 e6 2. d4 d5 Nc3 Bb4 4. e5 c5 5. dxc…, one of the commercial FRITZ books presents us the following data:

Move	#	%	Elo	performance
5. … Nc6	3	0	2505	1692
5. … Qc7	2	0	2512	1707
5. … Ne7	2	25	2548	2301

From a statistical point of view, the move 5. … Ne7 seems to be the strongest move, because it has been partially successful with 25% of the points and has been played by reasonable strong players with an average of 2548 Elo. Nevertheless, the chances for Black seem low, because even the strong players could only reach a performance of 2301 Elo with 5. … Ne7 in this position.

The fact that even the top book writers look at the statistical data may be astonishing. It is generally believed that Chess is dominated by strategic thoughts and, moreover, the fundamentals of statistics are severely violated in strategic games. Let us, e.g., assume that a program A plays some hundreds test games against another

program B. Let B score 75% of the points. Do we now know that B is stronger than A? No, we only know that B's score is not the result of a random process based on some normal distribution with a peak at zero. It is still possible that the programmer observes that A played only 1. e4 and 1. d4 and that A lost all d4-games and won all e4-games. Because A can in principle choose whether it plays 1. e4 or 1. d4, A is the stronger program.

Moreover, there are many examples in Chess in which a certain move in a certain position could be proven to be a bad move, whereas it was previously believed that the move was a good one. In these examples, the statistical information will look favorable for all cases, because the move will never be played again.

3 The New Way

We are convinced that strategic information is more valuable in games such as Chess than statistical knowledge plus medium-player expertise can be. The best experts for strategic information are the top chess programs themselves. Nevertheless, the statistical information seems to possess some value, because our experiments show that playing without any opening book is not a good option. Instead, we want to make statistical information available to the chess program as a second-order evaluation criterion for a move, during its search process. We distinguish between risk information that we want to use in order to spare time, and bonus information in order to make the program tend to play moves which have been successful in the past.

Let a big game database be available, and let us be interested in the value of a certain move in a specific position. From Section 2.2 we know that commercial tools may give information such as (a) how often a move has been played, and (b) how strong the players who played the move are on the average, (c) how successfully the move has been played. However, there is more information in the database. For instance, it makes a difference whether (a) a leading player (a so called Top-GM) played the move, and (b) whether the performance of the move is mainly based on draw games. Moreover, may we may consider (c) whether the move has been played in recent games after 1999, and (d) whether it has been played by especially aggressive players. Because good players tend to play weaker moves against weaker players (in order to hide their knowledge), we also distinguish between good and bad games. A game is estimated the better, the higher the two protagonists are ranked. So-called top games are games with both players having at least `BookTopElo` Elo. After all, the following eight parameters which influence a move's bonus and its risk are used for a weighted sum.

- `BookTopFac` = 8 — Weight of a Top-GM game.
- `BookDrawFac` = 4 — Weight for draw-game.
- `BookWinFac` = 8 — Weight for win-game.
- `BookRecentBias` = 1 — A move's weight increases, the more often it has been played after 1999.
- `BookTopBias` = 8 — Similar to `BookTopFac`, but independent of its results.

- `BookAggrBias` = 6 — A move is the better, the more often it has been played by an aggressive player (e.g., by Kasparov, Tal, Larsen, or Vaganian).
- `BookTopElo` = 2700 — A game is a top-GM game if both players of a game have at least `BookTopElo` Elo.
- `BookMinElo` = 2500 — Only games in that both players have at least BookMinElo Elo are of any interest.

The numbers assigned to the parameters are the values that we have used for experiments in Section 4.

Moreover, a 'spam'-filter suppresses games. If the words 'Blitz', 'Rapid', 'Blind', 'Interrnet', or '.com' occur in the `Event` or `Site` field of the pgn-game-header, we do not consider games with these keywords.

Let $\texttt{Games}_p\texttt{[i]}$ describe how often move `i` has been played in position `p`. $\texttt{TopGames}_p\texttt{[i]}$ is the number of games in that players with more than `BookTopElo` Elo have played move `i` in position `p`. $\texttt{Wins}_p\texttt{[i]}$ are defined as the number of games which have been won with move `i` in position `p`, $\texttt{Draws}_p\texttt{[i]}$ as the number of games which have ended in a draw with move `i` in position `p`. It is analogous with $\texttt{TopWins}_p\texttt{[i]}$, $\texttt{RecentGames}_p\texttt{[i]}$, and $\texttt{GamesAggr}_p\texttt{[i]}$. Let `Gamesum` be defined as follows.

$$\texttt{Gamesum}_p\texttt{[i]} := (\texttt{TopGames}_p\texttt{[i]}*\texttt{BookTopFac}+\texttt{Games}_p\texttt{[i]})$$

For a move `i` in position `p`, we define its goodness as:

$$\begin{aligned}
&\texttt{Goodness}_p\texttt{[i]} := (\\
&\quad \texttt{Wins}_p\texttt{[i]} * \texttt{BookWinFac} + \texttt{Draws}_p\texttt{[i]} * \texttt{BookDrawFac} && \text{(a)}\\
&\quad + \texttt{BookTopFac} * (\texttt{TopWins}_p\texttt{[i]}*\texttt{BookWinFac} +\\
&\qquad\qquad \texttt{TopDraws}_p\texttt{[i]}*\texttt{BookDrawFac}) && \text{(b)}\\
&\quad + \texttt{RecentGames}_p\texttt{[i]} * \texttt{BookRecentBias} && \text{(c)}\\
&\quad + \texttt{TopGames}_p\texttt{[i]} * \texttt{BookTopBias} && \text{(d)}\\
&\quad + \texttt{GamesAggr}_p\texttt{[i]} * \texttt{BookAggrBias} && \text{(e)}\\
&\;) \;/\; (\texttt{Gamesum}_p\texttt{[i]}), && \text{(f)}
\end{aligned}$$

and its risk as:

$$\texttt{risk}_p\texttt{[i]} := 10 \;/\; (3*\texttt{sqrt}(\texttt{Gamesum}_p\texttt{[i]})).$$

The goodness is then adjusted by

$$\texttt{Goodness}_p\texttt{[i]} \mathrel{+}= (\texttt{Gamesum}_p\texttt{[i]} *3) \;/\; \texttt{totalgames},$$

`totalgames` being the number of how often position p occurred in the database.

The intuition behind terms (a) to (f) is the following. Term (a) weights winning games and draw games against each other. Term (b) does the same, but here only the games of grandmasters with a high Elo rating count. Term (c) gives a bias to games which are played after 1999. Concerning term (d), top-games, i.e., games in that both players had at least 2700 Elo, should have higher impact than other games. In order to avoid openings which are preferred by calm players, games of Kasparov, Tal, Larsen and Vaganian are preferred with the help of term (e). Last but not least, the weighted sum is normalized in (f). The risk of a move `i` in a position `p` says that we can trust a move more, it has been played more often, always under the assumption that the program acknowledges the move as being 'good'.

Let us now assume that our program is in position `p`, and that the desired time for the next move is `m` minutes. Our program starts its computations as if there was no database in the background, but it adds the 'Goodness' to the root moves. This is achieved by appropriately changing the Alphabeta window at the root. If move `i` becomes the preferred move by the chess program itself, we will additionally adjust the computing time by the factor $risk_p[i]$. The new desired computing time is $m*risk_p[i]$.

4 Experiments

Finding convincing metrics and methods to evaluate a new idea, is quite a challenging task. Two important obstacles are: (1) if one runs experiments with a public-domain chess program, one will be in the danger that the program will not be sufficiently strong to fulfil the expert's requirements, and (2) practitioners may argue that results against weaker programs are irrelevant for top programs. Moreover, the whole idea has been developed as an attack against the top-book writers. Top books, however, are not publicly available. The first obstacle is not our problem, because we can make all experiments with our program HYDRA [6], which is strong enough to fulfill the requirements. The second obstacle is more severe. All that we can do is to present the data that we have: many test games against the top PC-chess program SHREDDER with a commercially available opening book, and some few games of 'real' competitions from public events.

4.1 The Program Hydra

The HYDRA project [6] is internationally driven and financed by PAL Computer Systems in Abu Dhabi, United Arabian Emirates. The core team consists of programmer Chrilly Donninger (Austria), researcher Ulf Lorenz (Germany), Chess Grandmaster Christopher Lutz (Germany), and the Pakistani project manager Muhammad Nasir Ali (Abu Dhabi). The FPGA chips from XilinX USA are provided on PCI cards from AlphaData in the UK. The compute cluster is built by Megware in Germany, supported by the Paderborn Center for Parallel Computing.

The hardware architecture. HYDRA uses the CHESSBASE/FRITZ graphical user interface (GUI), running on a Windows-XP PC. It connects via the Internet, using the

secure shell (ssh) to a Linux cluster, which itself consists of 8 Dual PC server nodes, being able to handle two PCI busses simultaneously. Each PCI bus contains one FPGA accelerator card. One MPI process is mapped onto each of the processors and one of the FPGA cards is associated with it as well. A Myrinet network interconnects the server nodes.

The software architecture. The software is partitioned into two parts: (i) the distributed search algorithm, running on the Pentium nodes of the cluster and (ii) the 'soft-coprocessor' on the Xilinx FPGAs, related to [5].

The distributed search algorithm: Similar to [7,8,15], the basic idea of the parallelization is to decompose the search tree in order to search parts of the search tree in parallel and to balance the load dynamically with the help of the work-stealing concept. First, a special processor P_0 receives the search problem and starts performing the forecast algorithm as if it would act sequentially. At the same time, the other processors send requests for work to other randomly chosen processors. When a processor P_i that is already supplied with work, catches such a request, it checks whether there are unexplored parts of its search tree ready for evaluation. These unexplored parts are all rooted at the right siblings of the nodes of P_i's search stack. P_i sends back either that it cannot serve with work, or it sends a work packet (a chess position with bounds etc.) to the requesting processor P_j. Thus, P_i becomes a master itself, and P_j starts a sequential search on its own. The processors can be master and worker at the same time. The relationship dynamically changes during the computation. When P_j has finished its work (possibly with the help of other processors), it sends an answer message to P_i. The master/worker relationship between P_i and P_j is released, and P_j becomes idle. It again starts sending requests for work into the network. When a processor P_i finds out that it has sent a wrong Alphabeta window to one of its workers P_j, it makes a so-called window-message follow to P_j. P_j stops its search, corrects the window, and starts its old search from the beginning. If the message contained a so-called cutoff which indicates superfluous work, P_j just stops its work. HYDRA achieves speed-ups of 18 on the 32 cluster entities [17].

The soft-coprocessors: At a certain level of branching, the remaining subproblems are small enough that they can be solved with the help of a 'configware' coprocessor benefiting from the fine-grain parallelism inside the application. There is a complete chess program on-chip, consisting of modules for the search, the evaluation, the generating of moves and doing or taking back moves. The three main advantages of our configware solution are as follows.

1. Implementation of more knowledge requires additional space, but nearly no additional time.
2. FPGA code can be debugged and changed like software, without the long ASIC development cycles.
3. A large amount of fine-grain parallelism can be used.

At present, we use 67 BlockRAMs, 9879 Slices, 5308 TBUFs, 534 Flip-Flops, and 18403 LUTs. The longest path consists of 51 logic levels, and the design runs at 50MHz. An upper bound for the number of cycles per search node is 9 cycles.

4.2 Results

We played three different matches with HYDRA 1.09 against SHREDDER 8.0, with 72 games in each match. Before each match, we erased the `Hydra.plr` and the `Shredder.plr` learning files in order to keep these influences fair over all experiments. In the first match, SHREDDER played with its commercial opening book, without any learning, and HYDRA played without any book at all. In the second match, HYDRA used the opening book heuristic, as described above, called 'autobook'. In the third match, HYDRA used the autobook, and SHREDDER used the 'optimal' parameters, including book-learning. We do not exactly know how SHREDDER's book learning works. One possibility is described in [4].

Table 1. Results of the experiments

Match	Outcome	Strength
HYDRA 1.09, no book vs. SHREDDER 8.0, commercial book, no book-learning	+27,–25,=20	+9 Elo
HYDRA 1.09, autobook vs. SHREDDER 8.0, commercial book, no book-learning	+37,–20,=15	+83 Elo
HYDRA 1.09, autobook vs. SHREDDER 8.0, commercial book, optimal parameters, full book-learning	+33,–21,=18	+58 Elo

In Table 1 we see a clear positive influence of our autobook. For instance, the first line +27,–25,=20 means that HYDRA won 27 of the 72 games, lost 25, and drawed 20. We were impressed sufficiently to test this heuristic in tournament games as well.

In an 8-game match against SHREDDER in Abu Dhabi, HYDRA scored 5.5 out of 8 points and won at least the first 2 games because SHREDDER's opening book was outperformed. Moreover, we have never had to experience that we came with a bad position out of the opening. The technique was also successful against humans. In Abu Dhabi, HYDRA could beat GM Vladimirov (2650 Elo) with 3.5 out of 4, and in the Man vs. Machine World Championship in Bilbao in 2004, HYDRA scored 3.5 out of 4 points against Topalov (2757 Elo), Ponomariov (2710 Elo, 2x), and Karjakin (2576 Elo) [9].

5 Conclusions

We proposed a new heuristic how to combine the chess expertise of a computer with partially dirty statistical information. The technique is so successful that it is now used in the leading world chess program HYDRA [6].

References

1. I. Althöfer. Root Evaluation Errors: How they Arise and Propagate. *ICCA Journal*, 11(3):55–63, 1988. ISSN 0920-234X.
2. D.F. Beal. An Analysis of Minimax. *Advances in Computer Chess 2* (ed. M.R.B. Clarke), pages 103–109, Edinburgh University Press, 1980.
3. M. Buro. The Othello Match of the Year: Takeshi Murakami vs. Logistello. *ICCA Journal*, 20(3):189–193, 1997. ISSN 0920-234X.
4. M. Buro. Towards Opening Book Learning. *ICCA Journal*, 22(2):98–102, 1999. ISSN 0920-234X.
5. J.H. Condon and K. Thompson. Belle Chess Hardware. *Advances in Computer Chess 5* (ed. D.F. Beal), pages 44–54, Pergamon Press, 1982.
6. C. Donninger and U. Lorenz. The Chess Monster HYDRA. *In Proc. of 14th International Conference on Field-Programmable Logic and Applications*, Antwerp, Belgium, LNCS 3203, pages 927–932, 2004.
7. R. Feldmann, B. Monien, P. Mysliwietz, and O. Vornberger. *Distributed Game Tree Search. Parallel Algorithms for Machine Intelligence and Vision*, pages 66–101, Springer-Verlag, 1990.
8. C. Ferguson and R.E. Korf. Distributed Tree Search and Its Applications to Alpha-Beta Pruning. In: *Proceedings of the 7th National Conference Artificial Intelligence*, pages 128–132, 1988.
9. H.J. van den Herik. The World Chess Team Championship Man vs. Machine. *ICGA Journal*, 27(4):246–248, 2004. ISSN 1389-6911.
10. F. H. Hsu. Large Scale Parallelization of Alphabeta Search: An Algorithmic and Architectural Study with Computer Chess. *PhD thesis*, Carnegie-Mellon University, Pittsburgh, 1990.
11. H. Iida, J.W.H.M Uiterwijk, H.J. van den Herik, and I.S. Herschberg. Potential Applications of Opponent-Model Search. Part 2: Risks and Strategies, *ICCA Journal*, 17(1):10–15, 1994. ISSN 0920-234X.
12. D.E. Knuth and R.W. Moore. An Analysis of Alphabeta Pruning. *Artificial Intelligence*, 6(4):293–326, 1975.
13. D.N.L. Levy. The 2nd Bilbao Man vs. Machine Team Championship. *ICGA Journal*, 28(4):254–255, 2005. ISSN 1389-6911.
14. D. Levy and M. Newborn. *How Computers Play Chess.* W.H. Freeman and Company, New York, 1990.
15. T.A. Marsland, A. Reinefeld, and J.Schaeffer. Multiprocessor Tree-Search Experiments. *Artificial Intelligence*, 31(1):185–199, 1987.
16. D.S. Nau. An Investigation of the Causes of Pathology in Games. *Artificial Intelligence*, 19(3):257–278, 1982.
17. Personal communication with S. Meyer-Kahlen, M. Feist, F. Morsch, A. Kure, E. Günes, C. Lutz, C. de Gorter, 2000–2003.
18. A. Plaat and J. Schaeffer. Kasparov vs. Deep Blue: The Rematch. *ICCA Journal*, pp. 95–101, 1997.
19. J. Schaeffer, R. Lake, P. Lu, and M. Bryant. Chinook: The World Man-Machine Checkers Champion, *AI Magazine*, 17(1):21–29, 1996.
20. Claude E. Shannon. Programming a Computer for Playing Chess, *Philosophical Magazine*, 41(314):256–275,1950.
21. J.W.H.M. Uiterwijk, H.J. van den Herik, and L. V. Allis. A Knowledge-Based Approach to Connect-4. The Game is Over: White To Move Wins. *Heuristic Programming in Artificial Intelligence I*, pages 113–133, 1989.