Rapier: A Chess Engine

A Final Report Presented to

Professor Rina Dechter

Natalia Flerova

University of California, Irvine

CS 175: Project in Artificial Intelligence

by

Team 13

Andrew Isbell

Christopher Pestano

Denise Ma

March 2010

1. Introduction

Rapier is the name of our chess engine. Our goal for this project was to make a chess engine that can compete at a high level against human chess players. Since we knew that ultimately we would be testing our engine on chess servers on the Internet, we were in a position to tailor our engine's evaluation function of a game position to succeed in our specific testing environment. While we did pit our engine against other chess engines, it was merely to satisfy our curiosity. Our criterion for success only required that our engine beat human players: We would be satisfied with the performance of *Rapier* if it performed at a level that, if a human were to perform at this level, he would receive the title "Chess Expert". We intended to make an "expert system".

Our criterion for success is defined in terms of the widely adopted Elo rating system. The Elo rating system is a method for calculating the relative skill levels of players in two-player games. Each player begins with a positive integer value as his rating. When two players play a game, the winner takes some of the loser's rating points and adds it to his own rating. The amount of points taken is a function of each of the players' ratings and experience: If a high rated player beats a low rated player, he only takes a small amount of his opponent's rating points, but if a low rated player beats a high rated one, he takes a large amount of rating points. A player who has only played a few games will have his rating fluctuate wildly, while a player who has played many games will have a more robust rating that is harder to change, for better or worse. The formulas are the following:

*Change = 1/(10^((HigherPlayer'sRating * SQRT(n1) – LowerPlayer'sRating * SQRT(n2))/ 2000)+1)*

Where n1 is the number of games the higher rated player has played and n2 is the number of games the lower rated player has played.

The computed value of Change is added to the winner's current rating and subtracted from the loser's current rating.

We had *Rapier* play games against human opponents on the Free Internet Chess Server <http://www.freechess.org/>. The matchmaking service used by FICS is designed to select opponents that have roughly the same Elo rating while preventing players from collaborating to increase a friend's rating artificially. The technical details of this matchmaking program are not available to us.

We are now in a position to state our criterion for success: After one hundred games played on the FICS, we will be satisfied with the performance of *Rapier* if it holds an Elo rating of at least 2000: if a human were to achieve this rating, he would be given the title "Chess Expert" by the World Chess Federation [2].

2. Background of Previously Done Research

In this section we will discuss previously done research that applies to our project on both computer chess and artificial intelligence.

2.1 Background of Computer Chess

The computer chess community lacks the organizational structure of academia. The best chess engines keep their algorithms, heuristics, evaluation functions, and techniques a close secret so that they can make a profit from selling their product. This makes it hard to list a handful of definitive sources on the topic, rather we learned about the progress of computer chess research from many articles, websites, and books. We will cover the most interesting and relevant points in this section.

We were first inspired to tailor our chess engine to beat humans after learning about the recent success of the best humans against the best computers. When Deep Blue defeated world champion Gary Kasparov in 1997, the moment became a symbol of the transition to the information age. However it is less widely known that since that time, top chess players have been faring much better against the best computer challengers. It is no longer the case that the best human chess players consistently lose to the top chess programs [3]. There is a trend of progress amongst world-class chess: it is certainly the case that the best human chess players at any given time throughout history have shown a steady improvement [3]. However this improvement cannot compete with the rate at which computational resources have been increasing so this cannot explain for the recent resurgence of the top human players against the computer challengers. Instead the reason for this recent success lies in a more thorough understanding of chess engines and how they play.

Most chess players have at best only an amateur understanding of game trees and no understanding at all of the details of algorithms like alpha-beta pruning that formally define the strategies of the computer chess engine. Intuitively speaking, the way to beat a chess engine that searches at a fixed depth of k levels in the search tree, one must make moves that lead to advantageous positions where the computer chess engine's evaluation function would only calculate the advantage if it had been able to search at least $k+1$ levels in the search tree. The way one would explain this technique to a chess expert differs from the way one would explain it to a computer scientist, and it was not until certain people with a good understanding of both fields stepped forward did the top human players gain the strategic weapons they needed to slay the cold, calculating chess engines of our modern times; it is no longer IBM's Deep Blue whom they must face. Instead even stronger engines with intimidating names such as *Shredder* and *Hydra* have appeared on the scene. These engines have a far greater number of professionals whose careers are specifically devoted to improving these engines [4].

But this project is not about beating computers, it is about beating humans! However the preceding information is still relevant to this report because it put us on a track to implement

techniques that take specific advantage of way a computer selects a move. Since the top chess engines who participate in these events keep their techniques a secret, they were of absolutely no help to us and we would have to design and implement these techniques ourselves. It is worth pointing out, however, that even if we had the source code of the top chess engines, it would not help us in creating an engine that took advantage of the specific strengths a computer has over a human. This is because the best chess engines are multi-purpose tools. They are not used solely for beating humans, as *Rapier* is. Instead they must also assist players in their study of the game – they help humans prepare for when they must face other humans. Furthermore they are often entered into competitions against other computer chess engines where anti-human heuristics would weaken their playing strengths.

The particular advantage unique to computer chess engines that we intend to capture is the following: tactical maneuvers whose culmination occurs within k moves are *always* found by the chess engine: it always finds its own short-term forcing moves and always avoids its opponent's short-term forcing moves. Furthermore strategic maneuvers whose culmination occurs in more than k moves are *never* found by the chess engine: it will never find a long series of forcing moves that lead to advantage and it will never see opponent's long series of forcing moves that lead to bad positions for the engine. k is the fixed depth of the search tree. [5]

Knowing this, it is now clear that the goal is to encourage the chess engine to reach positions that contain a large number of short-term tactical maneuvers. The computer will relentlessly punish the human opponent for any short-term mistake he makes while deftly evading all short-term threats presented by the opponent.

Our strategy to implement these techniques is to create an evaluation heuristic that quantifies this notion which we will name the "tacticity" of a position. If a numerical value that represents the saturation of the position with short-term tactical maneuvers can be easily obtained, we can find an appropriate weight to this value and use it directly in our evaluation function.

When thinking hard about how to quantify the tacticity of a position, we recalled the contributions of Tigran Petrosian, chess world champion 1963-1969. Petrosian was awarded the M. Phil degree from Yerevan State University, Armenia for his thesis "Chess Logic". His lasting contribution to chess strategy was his idea of "Overprotection": protecting ones pieces more than seems necessary in order to make the position *barren of* short-term tactical opportunities for your opponent. [6] Recall that computer chess engines thrive on positions that are *rife with* tactical opportunities. Overprotection is something that can be quantified. This can be done easily by counting the instances of a piece protected by another piece. A quick improvement is to scale each of these values by a factor inverse to the piece's value (this represents that one ought not to trade a queen for a knight, for example). The commensurability of overprotection allows us to capture the elusive notion of the "tacticity" of a position. It can be scaled by a constant negative factor before being added to the evaluation function. Because of the inverse relationship between overprotection and the tacticity of a position, we name this technique "Inverse Petrosian Evaluation".

2.2 Background of Relevant Artificial Intelligence Research

Our chess engine uses alpha-beta pruning heuristics for the minimax adversarial search. There is controversy as to when this was invented but one source claims McCarthy in 1956 first demonstrated this algorithm at the Dartmouth Conference.[7]

Linear regression is a static, supervised machine learning technique. This algorithm outdates the academic study of algorithms. It was created to assist bankers in the field of finance. [8] We use linear regression to learn the relative value of each of the chess pieces: pawn, knight, rook, queen, king. To do this we downloaded a database of chess positions, extracted how many pieces of each type each side had remaining, and which side went on to win the game. We used linear algebra techniques to find the optimal least squares equation. [9][10] We obtained the following results:

Piece	Weight
White Pawn	1.000
White Knight	2.126
White Bishop	3.4021
White Rook	5.0793
White Queen	10.2783
Black Pawn	-0.9976
Black Knight	-2.4666
Black Bishop	-3.3997
Black Rook	-5.1073
Black Queen	-9.7664

Due to the constraints of our underlying framework, we decided to average the values between opposing colors for each type of piece and scale each weight by 100. This yielded the following:

Piece	Weight
Θ_0: Pawn	100
Θ_1: Knight	230
Θ_2: Bishop	340
Θ_3: Rook	509
Θ_4: Queen	1002

The evaluation function using the values from this table has the following form where Q^w is the number of white queens and Q^b is the number of black queens, etc:

$$Evaluation = \Theta_5(K^w-K^b)+\Theta_4(Q^w-Q^b)+ \Theta_3(R^w-R^b)+ \Theta_2(B^w-B^b)+ \Theta_1(N^w-N^b)+ \Theta_0(P^w-P^b)$$

It can be seen that the evaluation function gives positive values to positions that favor white and negative values to positions that favor black. An evaluation of exactly zero indicates the position is equal for both players.

Our evaluation function contains further nuances but the evaluation as a whole has the same form as the one given above, just with more variables with different coefficients. The weights of the following components of the evaluation function were not machine learned. We found that static, unsupervised learning techniques were ineffective due to our inability to find a database that could give meaningful results when our regression algorithms were used upon it. We considered using reinforcement learning algorithms but as it will be seen in the Results section of this report we had to test *Rapier* manually which took a long time. We did not have enough time to have enough testing data for reinforcement learning algorithms to have meaningful results. The following components of our evaluation function were hard coded and ended up increasing performance more than any of our attempts to machine learn them.

Piece values are modified by other positional factors, for example a pawn on the left or right-hand side of the board is given a value of 85 instead of the typical 100 because it can attack only one direction. An isolated pawn – a pawn that does not have friendly pawns on either adjacent column – is also given a lower value from 80 – 88 depending on how close it is to the center of the board.

The Inverse Petrosian Evaluation outlined in the previous section is calculated as follows: For each instance of a piece defending a friendly piece, decrease (if *Rapier* is white) or increase (if *Rapier* is black) the rest of the evaluation by the following value:

*50 * (1/SQRT(ValueOfDefendingPiece))*

This results in *Rapier* evaluating positions that contain many unprotected pieces (for either side) as favorable and positions that contain many protected pieces (for either side) as unfavorable.

3. Tools

In our project we used the following tools: Eclipse, MATLAB, Microsoft Visual Studio, and Microsoft Office. We used Eclipse to write some Java helper functions such as extracting information from the database of chess positions. We used MATLAB to perform the linear algebra equations on the information after we extracted it from the database. The chess engine is written in C# and the programming environment we used for this is Microsoft Visual Studio. We used Microsoft Office software to create this report.

We are using the “Chess Engine Starter Kit” and graphic user interface found at the following website:

<http://chessbin.com/>

The Starter Kit comes with code for board representation, piece representation, and code defining how a piece. Nothing that resembles “Artificial Intelligence” is included: the search and the evaluation function were left to us.

4. Data

In our project we used the ChessX database of chess positions in Forsyth-Edwards notation: < http://chessx.sourceforge.net/>

5. Input/Output

Our chess engine adheres to the Chess Engine Communication Protocol. CECP is an open communication protocol that enables a chess engine to communicate with its user interface.

6. Results

6.1 Methodology and Data

We had the final version of *Rapier* play one hundred games on the Free Internet Chess Server. *Rapier* was selected randomly to play with the white or black pieces, and each side had five minutes to make all his moves in the game. If a player ran out of time when it is his turn he loses the game. “Five-minute blitz” chess using these rules is perhaps the most popular of ways chess is played on the Internet today.

Rapier recorded a score of +69-19=12: sixty-nine wins, nineteen losses, and twelve draws. *Rapier’s* Elo rating after one hundred games was 2063: exceeding our criterion for success.

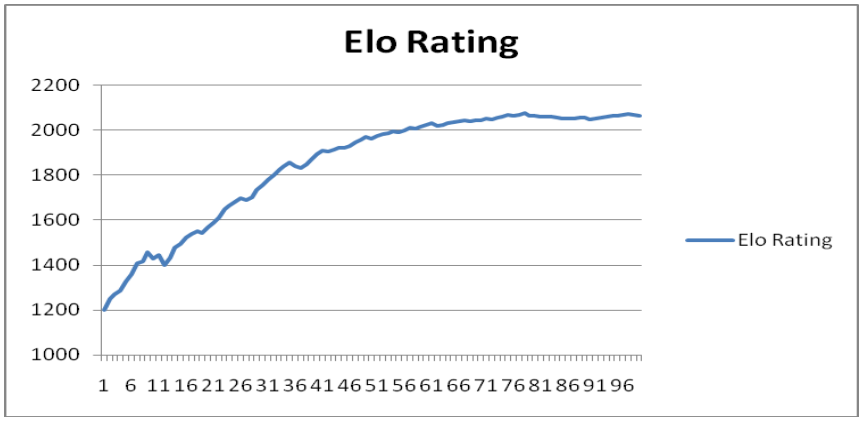

Fig 6-1

6.2 Analysis

Rapier recorded more of its wins during the beginning of the testing phase. At the end of the testing phase it won about as many games as it lost. Recall our description of the Elo rating system given in section 1 of this report: *Rapier* began with the initial Elo rating of 1200 and had to face low rated opponents who either just created their account or are genuinely of a weak playing strength. As you can see in figure 6-1 *Rapier's* Elo converged to roughly 2050 and finished with a rating of 2063. Because *Rapier* won about as many games as it lost during the final games of the testing phase and its rating converged at the end of the testing phase, this gives us confidence that *Rapier's* true playing strength is indeed above 2000.

7. Final Thoughts

We were extremely satisfied with *Rapier's* performance. Even though it took us ten hours to manually test one hundred games, it was an enjoyable experience to watch *Rapier* play chess. Amongst the memorable moments are a long initial winning streak, a close-fought game to break the 2000 point barrier for the first time, and the entertaining "trash talk" done on the part of our opponenets.

References

[1] Elo, Arpad 1978. *The Rating of Chessplayers, Past and Present*. Arco. ISBN 0-668-04721-6.

[2] World Chess Federation <http://www.fide.com/>

[3] Krabbe, Tim. "Defending Humanity's Honor"
<http://www.xs4all.nl/~timkr/chess2/honor.htm>

[4] Friedel, Frederick. "Hydra unbeatable in Abu Dhabi"
<http://www.chessbase.com/newsdetail.asp?newsid=1875>

[5]Friedel, Frederick. "A Short History of Computer Chess"
<http://www.chessbase.com/columns/column.asp?pid=102>

[6] Petrosian, Tigran. "Chess Logic" 1968. Yerevan State University. (Thesis for the M.Phil degree)

[7] McCarthy, John. 27 November 2006. "Human Level AI Is Harder Than It Seemed in 1955".
<http://www-formal.stanford.edu/jmc/slides/wrong/wrong-sli/wrong-sli.html>
[8] C.F. Gauss. 1821. *Theoria combinationis observationum erroribus minimis obnoxiae*.
[9] http://chessx.sourceforge.net/
[10] Björck, Åke. 1996. *Numerical methods for least squares problems*. Philadelphia: SIAM. ISBN 0-89871-360-9.

User Manual

Installing/Uninstalling Rapier:

1. Unzip the contents of the zip file.
2. Double click on setup.exe.
3. Follow the on-screen instructions to install the program.
4. The game will automatically launch once it is finished installing.
5. To uninstall, use the **Control Panel** to uninstall *Rapier*.

Starting and Playing the Game

1. The game is located in the **Team 13 [CS175]** folder through the **Start** menu.
2. The game initially starts with the AI disabled.
   - Enable this by clicking on the **Options** dropdown menu and checking **Enable AI**
3. The AI will always be the *black* player. Once the AI has been enabled, the black player will start responding to the white player's moves.
   - Note: if the AI is enabled while it's the black player's turn, the AI will not do anything. The AI has to be enabled before the white player performs a move.
4. Disable the AI at any time by un-checking **Enable AI** via the **Options** menu.
5. To move a piece, click on the piece and move it to positions that are highlighted.
   - Note: Positions will stay highlighted unless clicked pieces are clicked again, thereby "un-clicking" them.
6. To start a new game, click on **New Game** via the **File** menu. This will restart the game with the AI disabled.
7. Click on **File > Close** or click on the **X** button on the top right corner to close the program.

Hint: If the program isn't letting a piece move, then both your *King* is being checked and the move won't change it, or the move will result in your *King* being checked. There are no notifications for checks and checkmates!