Knowledge engineering in chess programming- Evaluation of a chess position-criteria and procedure

malugu_satyajit@yahoo.com
vidyuth_koniki@yahoo.co.in

Authors:

M.Satyajit
¾ Cse-A
Gitam

K.B.N.Vidyuth
¾ Cse-A
Gitam

Mr.Satyajit is a professional chess player; he is twice the captain of Andhra University. He is captain of the Gitam chess team for the last three years which won the hatrick-cup in the university meet. He represented the Andhra Pradesh state in several nationals and his current international rating is 2142.

Abstract:

Of all the board games chess has been the most difficult game to conquer (and so more interesting!) because it is not just sufficient to find a best move through search trees, we should also consider the recondite positional elements. Chess has some positional elements or some general principles of how to play a good game. These are very essential for assessing a move and/or position. In fact most of the human players do not think *that* deep (need not). They base their judgments depending on the position (positional judgment). They evaluate it according to some basic patterns. In this paper we show how a computer tries to do the same i.e. it identifies some patterns and then asses them in a Analysis function.

All the chess programs contain **Analysis function** that gauges the position depending on several constraints like material advantage and positional elements. Each piece on board is assigned some fixed values. Positional elements are also given values. (e.g.:- double pawn weakness can be given -0.25).

A chess programmer should have a thorough knowledge of various positional elements and he should find an efficient method of implementing this into his program. Hence chess becomes the combination of knowledge engineering and searching. Here the interesting aspect of the chess programming comes into picture.

- Knowledge- The opening and ending phases of the game can be played through predestined knowledge known as the ***'Book moves'***. For the middle game some patterns are identified by the programmer and given importance over others.
- Searching- Gaming trees which search the position for the best moves. Like ***Minimax*** and ***Alpha-beta pruning***.

An effective chess program must first identify the patterns on the board and assign values for each pattern. Some of these patterns include **double pawns, safety of the king, control of the center, space advantage, and mobility of pieces** which is given some values in the Analysis function. These elements are given points depending upon the situation and their importance.

What makes the chess program efficient is the artistic balancing of positional elements and the search techniques.

In this paper first we will introduce the *criteria* - positional elements in chess, and then the *procedure* we will show how to implement through analysis function them in a chess program. Then we will show the implementation of knowledge in the three stages of the game-**opening, middle game and endgame**. We will conclude with a criterion for testing how to test the efficiency of a program.

Paper:

Chess programming is unique field of AI research as it unifies two apparently different fields of research.

Searching- this is an age old field of research and still optimization is undergoing. The basic objective of chess is to find the best move in the position. For that from the basic Minimax to alpha beta pruning and several other improvements are made [1].

Knowledge- Rather than relying only on the computational power, it would be wise to use the knowledge of human players. It gives raise to an interesting aspect of chess programming of imparting knowledge to chess programs. Undoubtedly by doing so they become more powerful.

In fact strong grand masters never think as deep as a computer does, they base their assumptions on the evaluation of the position and experience. The positional sense of the player is the direct measure of his strength. The greatest romantic chess player of all times “Michael Tal” has a great positional sense, that though he calculates much less than his contemporaries he used to outmaneuver his opponents using his deep positional insight.

One of the most challenging tasks while creating a chess program is to convert the chess knowledge, which can be learned from many books, into computer code. Currently most computer chess programmers do this by explicitly coding many rules and afterwards weighting these rules. To successfully achieve this task, the programmer needs good positional chess knowledge and he shouldn't flinch from weeks of tedious optimizations.

In this paper we show the attempts made to evaluate a chess position using more human like approaches – pattern recognition, giving weights to different patterns,

Here we are not concerned with finding the best move on the board but with evaluation of a position. We also show how knowledge is imparted in the opening, middle and endgame stages.

Introduction to Evaluation:

The evaluation of a chess position is done based on two elements:

1. Material elements– all the pieces on the board are assigned values based on their power (mobility). King-4*, Queen-9, Rook-5, Bishop -3, knight -3, pawns-1.
2. Positional elements-various elements like pawn structure, mobility of pieces, king safety, space advantage etc are considered -not necessarily in the same order.

All the chess programs contain **Evaluation function** that gauges the position depending on these aspects. The very first task of this function is to count the material on the board, most of the times it is fairly equal so then we go to the positional elements.

This is the domain of knowledge so; it would be wise to have at least a rudimentary idea about the positional elements in chess. The understanding of the implementation needs a brief introduction to positional elements.

Positional elements:

Pawn structure:

The alignment of pawns (though these are of lowest value) is the most static and the most crucial element in a position.

As Tarrash put it “pawns are the soul of the game”.

Pawns are considered to be **weak** if

- Doubled or tripled pawns: Two or more pawns of the same color on the same file are usually bad, because they hinder each other's movement.
- Isolated pawns: A pawn which has no friendly pawns on either side is vulnerable to attack and should seek protection.
- Eight pawns: Having too many pawns on the board restricts mobility; opening at least one file for rook movement is a good idea
- More number of pawn islands: A cluster of connected pawns is said to be a pawn island. The more number of these islands the more vulnerable they are.

They are **strong** if

- Pawn chains: If all the pawns are in a chain fashion, they support themselves, block the opponent pieces.
- Passed pawns: Pawns which have advanced so far that they can no longer be attacked or rammed by enemy pawns are very strong, because they threaten to reach the back rank and achieve promotion.
- Mobile pawns: If the pawns have no immediate opposition, ie they are free to move at least a few squares then are strong.
- Pawn rams: Two opposing pawns or phalanxes- "butting heads" and blocking each other's forward movement constitute an annoying obstacle

Mobility of pieces:

Values are assigned to pieces basing on their ability to move on the board. Though these are fixed, they vary dynamically according to the position depending on the number of squares a piece can move in a given position. Mobility is direct measure of the activity of the piece. A knight on the edge of the board is weaker than the knight in the center of the board.

King safety:

Though all the other criteria in a position may be good they all will be futile if the king is vulnerable. So highest value should be given to king safety (sometimes even than the material advantage).

This is measured in “Tropism” of how easy it is for a piece to attack the opposing king, and is usually measured in terms of distance. The exact rules used to compute tropism vary by piece type, but they all amount to this: the closer you are to the opposing king, the more pressure you put on it.

Space Advantage:

We count the space advantage by counting the number of squares occupied in the opponent’s camp i.e. starting from the fourth rank we will just count the squares that are under the control of our pieces.

Implementation:

The standard of the evaluation function takes 1 as its basis for calculations (the value of a single pawn). Basing on this we give values to other pieces. A + sign indicates advantage for white and –ve indicates advantage for black. For example black is a knight ahead the evaluation function shows -3.00 {advantage of three *pawn units* for black}. Now for implementation of the positional elements we assign some values to each of these patterns. We list them as follows (these are not head and fast rules)

1. Doubled pawns are penalized -0.5 i.e. the two pawns summation will be only 1.5 not 2, Isolated pawns -0.25, Pawn islands -0.25 for each island
2. Mobile pawns +0.35 Passed pawn +0.25 its value increases as it rank progresses, Connected pawns +0.45
3. The more active a piece the more its value-one technique is to measure the piece mobility each time and give its value accordingly
4. Space advantage is generally given less value than other elements because is more volatile. It's never more than 0.50.
5. King safety is a very critical element. Our aim in the game is to check- mate the king. So if we are oblivious of it we may loose the objective and if more are concerned on it we cannot explore deep into elements which brings us victory eventually.

Now that we identified the basic and simple features we give the list of various other patterns

- Rooks on the seventh rank are encouraged if two rooks even more value is given
- If the position is open +.50 for pair of bishops
- If the position is closed +.35 for pair of knights
- Additional value for rooks on the open file.
- Additional value for speed development of pieces in the opening (below 10 moves). Some times we can evaluate development more than a cost of a pawn(s).
- Add for control of the central squares (e4,d4,e5,d5)
- Penalize for weak squares add points if opponents weak squares can be exploited.
- In endgame a passed pawn backed by a rook is counted as an asset.
- Penalize if King is with out the pawn cover

We can go on adding several constraints like these depending upon the position. In chess the major problem is there are no head and fast rules you have to change your assumptions according to the position. (You cannot go for an exploiting double pawn when your king is in danger). So implementing these patterns in a logical way is an immensely difficult task and hence chess programming, along with chess can also be considered as an ART. It has been calculated that there are hundreds of some other constraints. It is impractical to try to implement these in a program overnight in a single edition. Different chess programs (knight cap, fritz, shredder, rookie) use different allocation of values and we can truly idealize any particular program.

All these largely depend upon programmers taste. Hence programming chess unlike other areas of AI requires a thorough understanding of the intricacies and subtle points of its domain. The higher his understanding of the game the correct he can identify different elements and assign accordingly. In fact there is no perfect evaluation system. We experiment with various values and test it with the older edition, and then release the next edition if the new ideas increase its strength.
Unless you are after the World Championship, your evaluator does not need to be all-powerful!
Identification of the above patterns (doubled pawns, pair bishops…) is usually done by bit boards. Bit boards are a wonderful method of speeding up various operations on chessboards by representing the board by a set of 64 bit numbers.

The game of chess is divided into three parts:

1. Opening
2. Middle game
3. Endgame

Opening:

Just as human players memorize opening moves, Chess programs have opening books which is a database of stored moves and positions. This allows the Chess program to move instantly when a move is available in the book. Needless to say, this is a very great advantage that allows the program to save time on thinking and also it reduces fallibility.
One more great advantage of these opening books is that we can instruct a computer to play according some standard ideas relevant to an opening. We can also keep our program up to date with the latest developments in the international arena and be better equipped against the human players.
In chess opening literature there is a standard called Encyclopedia of Chess Openings this helps the program to arrange its database in a hierarchical order and also meaningful the user. This information is stored as a hash table on disk and can be looked up quickly.

Middle game:

Modern chess game databases are so vast that they have games of order 15,000,000 lakhs. Some programs refer to these databases and play accordingly to the extent where opponent also plays the same moves (only when the game is from authoritative source). This reduces the burden of calculations. So the programs are getting more and more powerful with the databases attached to them. Deep blue, which had a database of almost all the games played on the earth till date, played the first 22 moves perfectly in its last game of its match against Kasparov without much thinking and eventually won the match. This is the power of knowledge.

Endgame:

We now consider how the endgame database is included in the chess program.
Chess endgame tablebases (EGTBs) are special databases that store all possible positions with a given material balance and their results. The most popular among them is Nalimov. Each endgame tablebase file allows the user to check what the result is with best play. Most chess engines don't even need to reach the position covered in an endgame tablebases to use the tablebases. For example, a few moves before such a position, the engine calculates (but does not play yet) a series of exchanges that leads

directly to a position in the tablebase. The engine will then look up (or probe) the tablebase and get the results for that hypothetical position. This should help improve play of course and reduce the time of computing and sometime even computing altogether is avoided. These table bases are generated by induction.

Criteria for testing a chess program:

1. It should be able to beat the original program at least in 65% cases
2. Increased rate of defeating a human player
3. It should prove its rating in the online internet chess clubs like ICC, play chess, CERN.

Conclusion:

Chess has been described as the *Drosophila Melanogaster* of artificial intelligence, in the sense that the game has spawned a great deal of successful research (including a match victory against the current world champion and arguably the best player of all time, Gary Kasparov), much like many of the discoveries in genetics over the years have been made by scientists studying the tiny fruit fly.

Due to the high computing power of up to date processors chess programs are able to think ahead four and more moves even with short thinking time. Considering that, they are tactically far superior to even very strong human chess players. Nevertheless, the best human chess players are still able to occasionally win against the strongest chess programs. The reason for this is that a chess program has only a relatively modest positional knowledge compared to a world class chess player. Efforts have been still made in this direction to conquer the world of chess. Now this has become the domain of knowledge engineering rather than searching. If chess programs really were able to grab the world crown officially it would be the greatest achievement of the Artificial Intelligence. And knowledge engineering has a very big role in it.

Bibliography:

1. Artificial Intelligence procedures used in Chess Programming-By *M.Satyajit and K.B.N.Vidyuth*

2. http://www.chessprogrammingtheory.com

3. http://www.chessbox.de/Compu/schachwert3_e.html

4. http://www.aarontay.per.sg/Winboard/egtb.html

http://www.npac.syr.edu/copywrite/pcw/node341.html