IMPROVISING THE PERFORMANCE OF A CHESS ENGINE BY FINE TUNING THE EVALUATION FUNCTION

[1]G. V. S. Prasad, [2]P. Ch. N. V. Sairam, [3]M. Bhavana Sri, [4]M. Siva Krishna
[1] Associate Professor, [2,3,4] Student
CSE Department, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur Dt, AndhraPradesh,
gvs@kluniversity.in, sairam1390040@gmail.com, bhavanasri444@gmail.com, krishnasiva6745@gmail.com

ABSTRACT

Here, we propose a new feature set along with modified weights for evaluating the current chess board position. Most of the previous work adopted in this research area has normally adopted the co-evolution (i.e., tournaments among the virtual players) to decide which players will pass to next generation, depending on the outcome of each game. We considered a chess engine which reached an ELO rating of 2404 point for the best virtual player with supervised learning and a rating of 2442 points for the best virtual player with unsupervised learning. By adopting our feature set, we are claiming that our considered chess engine will reach to a rating of 2414 points with supervised learning and a rating of 2450 points for the best virtual player with unsupervised learning. We are proposing a method for improvising the performance of the above considered chess engine by tuning the evaluation function through new feature set which can be obtained by making some adjustments to the weights of the chess pieces and features in the chess engine. Finally, it is appropriate to mention that the performance of any chess engine can be enhanced by refactoring the evaluation function.

Keywords: ELO rating, chess board position, Evaluation function, unsupervised and supervised learning.

INTRODUCTION

During 1980s there is a huge interest towards developing some robots or computers to play computer vs human being games. Among all the games, Chess are considered as most enthusiastic game because chess is not only one of the most popular games all over the world, but also due to the reason that it is a game requiring high intelligence. Since the start of the century mathematicians have attempted to develop a software models to make master simulated players for a chess game. A lot of research work has been carried out in this area, where the most of the work is concentrated on producing the grand master level pc programs. One of the great invention in this area was a Deep Chess which is developed by IBM which had played against Garry Kasparov, the World Chess Champion and the deep blue defeated him through analyzing his moves which were stored in the deep blue database. Almost all chess knowledge of chess programs is defined in the chess evaluation function. This knowledge is presented with many arithmetic expressions and parameters.

PC, "Dark Blue", in 1997. "Dark Blue" and since "More profound Blue", still principally depend on best constrain strategies to pick up preference over the rival by inspecting further into the game tree. Be that as it may, as it is infeasible to assess the majority of the potential amusement ways of chess, we can't depend on best compel techniques alone. We have to grow better methods for approximating the result of amusements with assessment capacities. The computerized learning of assessment capacities is a promising exploration territory in the event that we are to deliver more grounded manufactured players.

For the past 50 years the stress has been on improving the heuristic search techniques and evaluation functions. The evaluation function yields a static evaluation which is then employed in the search algorithm. Therefore the evaluation function is the most important part of a chess program. Efficiency of a chess program depends on evaluation function speed and discovered knowledge. Evaluation function contains a lot of expressions and weights which were both formed by human experts. But because discovering optimal weights is very difficult we need a better way to find them. One possible method is automated tuning or “learning”. When we talk about automated tuning in computer chess we keep focus on algorithms such as hill climbing, simulated annealing, temporal difference learning and evolutionary algorithms. All approaches enable tuning on the basis of program’s own experiences or final result of chess games competition: win, loss, or draw.

In 1949, Shannon began to construe how PCs could play chess. He proposed the possibility that PCs would require an assessment capacity to effectively contend with human players. In spite of the fact that Shannon spearheaded the work on PC chess, it is Turing who is authorize with delivering the main chess machine in 1952.The most punctual production that effectively utilizes learning was exhibited in 1959 by Samuel. Samuel built up a checkers program that endeavored to locate "the most noteworthy point in multidimensional scoring space" by utilizing two players. The outcomes from Samuel's investigation were noteworthy but then his thoughts remained to some degree undeveloped for a long time.

In 1988, Sutton developed Samuel's considerations further and figured methodologies for 'common refinement learning' (TDL). Various examiners have since associated TDL to amusements [2, 3, 15, 16]. A champion among the best of these is Tesauro's backgammon program "TD-Gammon" which finished pro level status .Tesauro had shown that TDL was a serious device in the headway of world class diversions programming. Thrun adequately associated TDL to his program "Neuro Chess". TDL is explored in Kaebling . Co-transformative strategies try to build up a people of good contender game plans from potentially poor basic masses by "introducing the understudy in a learning space which responds to its own improvements in an unending winding" (Pollack). Moriarty gives a diagram into general learning with transformative estimations. In 2000, Chellapilla and Fogel successfully made methods for playing checkers with the usage of a people of neural framework candidate players. Barone associated adaptable making sense of how to convey a not too bad poker player. It was entered in a general rivalry including a couple of human ace players and fulfilled a situating of best 22%. Diverse instances of co-progression fuse Pollack's Backgammon player and Seo's change of frameworks for the iterated prisoner dilemma. Top PC chess programs are develop typically as for manual component decision and tuning of their appraisal work, generally through years of experimentation.

Different conveyed examinations revealed that distinctive investigators have attempted to develop a program that makes sense of how to play subjective preoccupations, given basically no earlier finding out about the lead of the diversion. A common chess playing machine by and large examines the moving possible results from a chessboard configuration to pick what the accompanying best move to make. The mammoth drive looking framework used by the Deep Blue chess engine has had tremendous impact in the ground of electronic thinking, yet in the meantime saw to be resource hungry. It's been just about quite a while since IBM's Deep Blue supercomputer beat the preeminent world chess champion, Gary Kasparov, all of a sudden under standard rivalry rules. Starting now and into the foreseeable future, chess-playing PCs have ended up being on a very basic level more grounded, leaving the best individuals insignificant shot even against a propelled chess engine running on a mobile phone. Nevertheless, while PCs have ended up being speedier, the way chess engines work has not changed. Their vitality relies upon monster drive, the route toward looking for through all possible future moves to find the best next one. Clearly, no human can facilitate that or come wherever close.

While Deep Blue was looking through somewhere in the range of 200 million positions for each second, Kasparov was most likely looking through close to five a moment. But he played at basically a similar level. Obviously, people have a trap up their sleeve that PCs still can't seem to ace. This trap is in assessing chess positions and narrowing down the most beneficial roads of pursuit. That drastically streamlines the computational errand since it prunes the tree of every conceivable move to only a couple of branches .Computers have never been great at this, yet today that progressions on account of crafted by Matthew Lai at Imperial College London. Lai has made a computerized reasoning machine called Giraffe that has shown itself to play chess by assessing positions considerably more like people and in a completely unique approach to ordinary chess motors. Straight out of the crate, the new machine plays at an indistinguishable level from the best regular chess motors, a considerable lot of which have been tweaked over numerous years. On a human level, it is equal to FIDE International Master Status, putting it inside the main 2.2 percent of competition chess players.

PREVIOUS WORK

Now we will discuss the previous work done in this field. First we will start from the IBM Deep chess, which is a chess program developed to play with the human and it is first played against Kasparov, world chess champion and the deep blue won the first game in six-game-match and finally Kasparov defeated the deep blue by 4-2.after some period the deep blue is modified and updated to a new version and again a match is played between Kasparov and deep blue. This time deep won the six-game-match.

To begin with, we will quickly talk about works that don't make utilize of an evolutionary algorithm for tuning the weights of the evaluation function of a chess engine. Thrun developed up the program Neuro Chess which learned how to play chess from final results with an evaluation function by neural systems. This work also included both temporal difference learning and explanation-based learning. Hsu et al. tuned the weights of their evaluation function for the computer Deep Thought (later called Deep Blue) utilizing a database of grandmaster-level games. Beal and Smith determined the values of the chess pieces of a chess game utilizing temporal-difference learning. In further work, the idea of piece-square values was introduced into their work.

Evolutionary algorithms have also been utilized before for tuning the evaluation function of a chess engine. Kendall and Whitwell utilized them for tuning the evaluation function of their chess program. Nasreddine et al. proposed a genuine coded evolutionary algorithm that incorporated the purported "dynamic boundary strategy" where the boundaries of the intervals of each weight are dynamic.

Boˇskovi´c presented three works that make the usage of differential evolution to change the weights of the evaluation function of their chess program. In their first work, they tuned the chess material values and the mobility factor of the evaluation function. The weights that obtained are matching with the values known from chess theory. In a second work, Boˇskovi´c et al. utilized adaptation and opposition-based optimization mechanisms with co-evolution to enhance the rating of their chess program. In a third work, Boˇskovi´c et al. enhanced their opposition-based optimization mechanisms with a new history mechanism which utilizes an auxiliary population containing competent individuals. This mechanism guarantees that talented individuals are retained during the evolutionary process.

Genetic programming is also been utilized for tuning the weights of the chess evaluation function. Hauptman and Sipper had evolved the strategies for playing chess end-games. Their evolved program could draw against the CRAFTY which is one state-of-the-art chess engine having a rating of 2614 points. In a second work, Hauptman and Sipper advanced whole game-tree searching algorithms to solve mate-in-N problems in which the opponent can't stop from being mated in at most N moves. It is worth for noticing that this work does not use the alpha-beta pruning algorithm.

Evolutionary programming has been used before by many other authors for tuning the weights of the chess evaluation function. Fogel et al. used this sort of evolutionary algorithm to improve the ELO rating of a chess program by 400 points. They tuned the material values of the pieces, the piece-square values, and the weights of the three neural networks. Their computer program has learned chess by playing the games against itself. In a second work, Fogel et al. integrated the co-evolution, but this time, they evolved their program during 7462 generations, reaching an ELO rating of 2650. The resultant program was called Blondie25. In a third work, Fogel et al, used rules for managing the time allocated per moveinside their program Blondie25. They attained a rating of 2635 points against the program Fritz8.0, which was rated # 5 in the world. It is noteworthy that Blondie25 was also the first machine learning based chess program that was able to defeat a human chess master.

Vazquez-Fernandez et al. used typical database chess problems to change the weights of the evaluation function of their chess engine. Using evolutionary programming as their search engine, they mutated only those weights that involved in the solution of the current problem and used the mutation mechanism through the number of problems solved by each virtual player. Using their algorithm they got the “theoretical” values of the pieces and gained an increase in the strength of their chess engine by 335 points. In a further paper, Vazquez-

Fernandez et al. used a database of the games played by the chess grand masters to change the weights of the material values and the mobility factor of the chess pieces. In this case, they got the "theoretical" values of the pieces with their evolutionary algorithm.

CHESS ENGINE

We considered the chess engine developed by Eduardo Vazquez Fernandeza mentioned in the paper "An evolutionary algorithm with a history mechanism for tuning a chess evaluation function" with the following characteristics:

• The search depth adopted by our engine is of one ply (which corresponds to the movement of one side) for the training phase as used in, and of six ply for the games among the virtual players recommended in.

• We used the alpha-beta pruning search algorithm.

• We incorporated a mechanism to stabilize positions through the Quiescence algorithm, which takes into account the exchange of material and the king's checks.

• We used hash tables and iterative deepening. A Hash Table is a data structure that allows saving the value of a given position on the board with the purpose of not having to calculate it again. Furthermore, chess engines do not perform your search with the alpha-beta algorithm to a fixed depth; instead they use a technique called iterative deepening. With this technique a chess engine searches at a depth of two, three, four, and so on, with the purpose that after finishing the search time, a catastrophic movement does not happen.

Our considered chess program evaluates the position of the virtual player A with the following expression:

$$eval = \sum_{i=1}^{22} Weight_i * f_i$$

where Weight_i is one of the weights shown in Table 1, and fi is the feature of the weight Weight_i for the virtual player A. The descriptions of the features are the following:

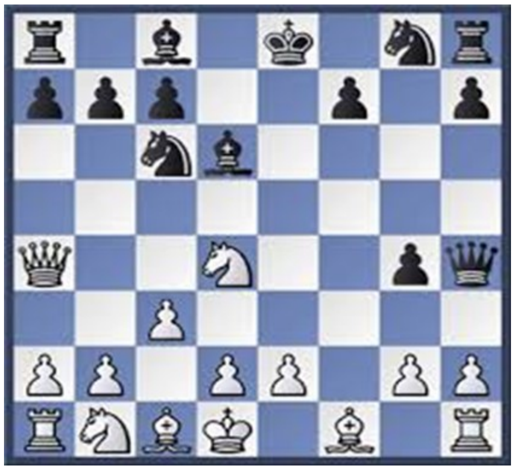

Figure 1: Example chess diagram to explain feature extraction

• fKING LONG CASTLING is a binary value. It is true if the king is castled along queen side;   otherwise, it is false. For example, in Fig. 1 this value is false for the white king.

• fKING SHORT CASTLING is a binary value. It is true if the king is castled along queen side; otherwise, it is false. For example, in Fig. 1 this value is false for the white king.

• fPIECE OPEN CHECK is a binary value. It is true if the PIECE is in front of king move directions and opponent power should be in that direction; otherwise, it is false. For example, in Fig. 1 this value is false for the white king.

• fROOK DOUBLED is a binary value. It is true if two rooks are in same column; otherwise, it is false. For example, in Fig. 1 this value is false.

• fQUEEN PINNED is a binary value. It is true if removal of any piece in front of queen leads to loss of a queen, otherwise, it is false. For example, in Fig. 1 this value is false.

• fPAWN VALUE is the number of pawns of the virtual player A. For example, in Fig. 1 the white player has six pawns.

• fKNIGHT VALUE is the number of knights of the virtual player A. For example, in Fig. 1 the white player has two knight.

• fBISHOP VALUE is the number of bishops of the virtual player A. For example, in Fig. 1 the white player has two bishop.

• fROOK VALUE is the number of rooks of the virtual player A. For example, in Fig. 1 the white player has two rooks.

• fQUEEN VALUE is the number of queens of the virtual player A. For example, in Fig. 1 the white player has one queen.

• fPAWN DOUBLED PENALTY VALUE denotes the number of doubled pawns of the virtual player A. For example, in Fig. 1 the white player has zero doubled pawns.

• fPAWN ISOLATED PENALTY VALUE denotes the number of isolated pawns of the virtual player A. For example, in Fig. 1 the white player has Zero isolated pawns.

• fPAWN BACKWARD PENALTY VALUE denotes the number of backward pawns of the virtual player A. For example, in Fig. 1 the white player has zero backward pawns.

• fPAWN PASSED denotes the number of passed pawns of the virtual player A. For example, in Fig. 1 the white player has zero passed pawns.

• fPAWN CENTRAL denotes the number of central pawns of the virtual player A (pawns located on squares c4, c5, d4, d5, e4, e5, f4 or f6).For example, in Fig. 1 the white player has zero central pawn.

• fKNIGHT SUPPORTED denotes the number of knights (of the virtual player A) supported or defended by one of his pawns. For example, in Fig. 1 the white player has one knights supported.

• fKNIGHT OPERATIONS BASE denotes the number of knights (of the virtual player A) in an operation's base (it is when a knight cannot be evicted from its position by an opponent's pawn). For example, in Fig. 1 the white player has zero knights in a operation's base.

• fKNIGHT PERIPHERY 0 denotes the number of knights (of the virtual player A) in the squares a1, . . ., a8, b1, . . ., g1, h1, . . ., h8, and b8,. . ., g8. For example, in Fig. 1 the white player has one knight in periphery 0.

• fKNIGHT PERIPHERY 1 denotes the number of knights (of the virtual
player A) in the squares b2, . . ., b7, c2, . . ., f2, g2, . . ., g7, and c7,. . ., f7. For example, in Fig. 1 the white player has zero knights in periphery 1.

• fKNIGHT PERIPHERY 2 denotes the number of knights (of the virtual player A) in the squares c3, . . ., c6, d3, e3, f3, . . ., f6, and d6, . . .,e6. For example, in Fig. 1 the white player has zero knights in periphery 2.

• fKNIGHT PERIPHERY 3 denotes the number of knights (of the virtual player A) in the squares d4, e4, d5, e5. For example, in Fig. 1 the white player has one knight in periphery 3.

• fROOK OPEN COLUMN denotes the number of rooks (of the virtual player A) in an open column (a column without pawns). For example, in Fig. 1 the white player has zero rooks in a open column.

• fROOK SEMIOPEN COLUMN denotes the number of rooks (of the virtual player A) in a semi-open column (a column that contains only opponent's pawns). For example, in Fig. 1 the white player has zero rooks in a open column.

• fROOK CLOSED COLUMN BEHIND denotes the number of rooks (of the virtual player A) in a closed column behind of its pawns. We defined a closed column as the column that contains pawns of both players. For example, in Fig. 1 the white player has two rooks in a closed column behind of its pawns.

• fROOK CLOSED COLUMN AHEAD denotes the number of rooks (of the virtual player A) in a closed column ahead of its pawns. For example, in Fig. 1 the white player has zero rooks in a closed column ahead of its pawns.

• fROOK SEVEN RANK denotes the number of rooks (of the virtual player A) in the seventh rank. For example, in Fig. 1 the white player has zero rooks in the seventh rank.

• fKNIGHT MOBILITY is the number of knights' moves of the virtual player A. For example, in Fig. 1 the knight's moves for the white player are 6 on d4.

• fBISHOP MOBILITY is the number of bishops' moves of the virtual player A. For example, in Fig. 1 the bishop's moves for the white player are 0.

• fROOK MOBILITY is the number of rooks' moves of the virtual player A. For example, in Fig. 1 this value is 0 for white rook on a1.

• fQUEEN MOBILITY is the number of queens' moves of the virtual player A. For example, in Fig. 1 the queens' moves for the white player are 8.

• fKING MOBILITY is the number of king's moves of the virtual player A. For example, in Fig. 1 the king's moves for the white player are 2.

• fKING ATTACKING MATERIAL is the sum of the material value of the pieces that are attacking the opposite king. We mean those pieces whose movements act on its opposite king's square or on its opposite king's adjacent squares.

• fKING DEFENDING MATERIAL is the sum of the material value of the pieces that are defending its king. We mean those pieces whose movements act on its opposite king's square or on its opposite king's adjacent squares.

• fKING CASTLING is a binary value. It is true if the king is castled; otherwise,
it is false. For example, in Fig. 1 this value is false for the white king.

• fKING PAWNS is the number of pawns located on its king's adjacent squares. For example, in Fig. 1 this value is two for the white king.

• fBISHOP AHEAD is the number of pawns which are in front of its bishop and obstructing its movement. For example, in Fig. 1 this value is two for the black bishop on c1.

• fBISHOP PAWNS MOBILITY is the number of movements of the pawns which obstruct the movement of the bishop. For example, in Fig. 1 this value is two for the black bishop on c1.

• fBISHOP PAIR is a binary value. It is true if the player A has the bishop pair; otherwise, it is false. For example, in Fig. 1 this value is true for the black and white side.

By using the above formula we can find out the average weights.

Table 1: Ranges of weights considered in our approach

Number	Weight	W_{low}	W_{high}
1	PAWN VALUE	50	50
2	KNIGHT VALUE	100	200
3	BISHOP VALUE	100	200
4	ROOK VALUE	200	350
5	QUEEN VALUE	400	700
6	PAWN DOUBLED PENALITY VALUE	-50	50
7	PAWN CENTRAL	-60	100
8	PAWN ISOLATED PENALTY VALUE	-60	60
9	PAWN BACKWARD PENALTY VALUE	-60	60
10	PAWN PASSED	-60	100
11	KNIGHT SUPPORTED	-60	110
12	KNIGHT OPERATIONS BASE	-60	60
13	KNIGHT PERIPHERY 0	-40	40
14	KNIGHT PERIPHERY 1	-40	40
15	KNIGHT PERIPHERY 2	-40	40
16	KNIGHT PERIPHERY 3	-40	40
17	ROOK OPEN COLUMN	-50	40
18	ROOK SEMIOPEN COLUMN BEHIND	-50	60
19	ROOK CLOSED COLUMN	-50	50
20	ROOK SEVEN RANK	-45	45
21	KNIGHT MOBILITY	0	100
22	BISHOP MOBILITY	0	100
23	ROOK MOBILITY	0	100

24	QUEEN MOBILITY	0	200
25	KING MOBILITY	0	50
26	ROOK SEMIOPEN COLUMN AHEAD	-50	50
27	KING ATTACKING MATERIAL	-90	10
28	KING DEFENDING MATERIAL	0	110
29	KING CASTLING	0	90
30	KING PAWNS	0	100
31	BISHOP AHEAD	-90	10
32	BISHOP PAWNS MOBILITY	0	100
33	BISHOP PAIR	0	110
34	KING LONG CASTLING	0	150
35	KING SHORT CASTLING	0	200
36	KING OPEN CHECK	0	100
37	ROOK DOUBLED	0	150
38	QUEEN PINNED	0	100

We considered the evolutionary algorithm given by the Fernandez is shown in Algorithm 1.

Algorithm 1. EvolutionaryAlgorithm()

```
1: intializePopulation();
2: g = 0;
3: while g <Gmax do
4:      scoreCalculation();
5:      selection();
6:      updateHistoricalMechanism();
7:      for i = N/2→N−1 do
8:          if 100 * rand(0, 1) <Pr then
9:              VP[i]←historicalMechanism();
10:         else
11:             VP[i]←mutate(VP[i-N/2]);
12:         end if
```

```
13:       end for
14:       g++;
15: end while
```

• The first step initializes the weights of N virtual players with random values within their corresponding boundaries.

• In the second step, we sets the generations counter equal to zero.

• In the third step (from the lines 3–15) we carry out the tuning of the weights for the N virtual players during Gmax generations. The description of the third step i.e., from line 3 to line 15 in which the line 4, calculates the scores of the virtual players in which the score of a virtual player is incremented by one when the movement of the virtual player for the current chess board position is similar to the database for which the virtual player did the same action as the human chess master. In the line 5 selection mechanism is carried out in such a way that only the best n/2 (half virtual players supplied) pass to the next generation. In line 6 an array of best virtual players which left the evolutionary process is maintained by updating the historical mechanism of virtual players. In lines 7–13 we obtain the second half of virtual players needed. If 100 * rand(0, 1) <Pr (where Pr is a control parameter defined by the user), we obtain the virtual player i from the historical mechanism, and if not, we mutate the virtual player i−N/2 to obtain the virtual player i. Finally, line 14 increases the generation counter in 1.

The procedure for computing the score of each virtual player is described in Algorithm 2.

```
Algorithm 2. scoreCalculation()
1: for i= 0→N−1  do
2:    score[i] = 0;
3: end for
4: for each position p in database S  do
5:    m = grandmasterMovement(p);
6:    setPosition(p);
7:       for each virtual player i    do
8:        n = nextMovement(i);
9:          if m = = n  then
10:            score[i]++;
11:         end if
```

```
12:     end for
13: end for
```

In lines 1–3, we establish the score counter to zero for each virtual player. Line 4 chooses p training positions from database S. Line 5 chooses chess grandmaster movement for position p. Line 6 sets the position p (this allows to each virtual player to calculate its next movement). Finally, each virtual player calculates its next move n, and if this movement matches movement m, this virtual player increases its score by 1. By using the above mentioned algorithms after running for certain number of iterations we can able to develop the chess engine with the mentioned ELO rating.

CONCLUSION

We have succeeded in improving the overall performance of a given chess engine by considering the new feature set. The previous work in this domain proved that evaluation function is the most critical component of any chess software system. We achieved this by considering the new different features in the evaluation function.

We would like to thank the chess domain experts who gave valuable suggestions in extracting the features from board positions taken from grand master games. We encourage the upcoming researchers to experiment more on identifying additional features. This can eventually lead to a powerful chess engine.

REFERENCES

1. Eduardo Vazquez-Fernandez, Carlos A. Coello Coello, Feliu D. Sagols Troncoso, “An evolutionary algorithm with a history mechanism for tuning a chess evaluation function”.
2. B. Boˇskoviˊ c, S. Greiner, J. Brest, V. ˇZumer, “A differential evolution for the tuning of a chess evaluation function”, in: 2006 IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, July 16–21, IEEE Press, 2006, pp. 1851–1856.
3. D.B. Fogel, T.J. Hays, S.L. Hahn, J. Quon, “Further evolution of a self-learning chess program”, in: Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05), Essex, UK, April 4–6, IEEE Press, 2005,pp. 73–77.

4. A. Turing, “Digital Computers Applied to Games, of Faster than Thought”, Pitman, 1953, pp. 286–310 (Chapter 25).
5. F.Hsu, T.Anantharaman, M.Campbell, A.Nowatzyk, “Deep thought, in: Computers, Chess and Cognition”, Springer, Berlin, 1990, pp. 55–78 (Chapter 5).
6. R. Hunter, “MM algorithms for generalized Bradley-Terry models”, The Annals of Statistics 32 (2004) 2004.
7. Tom M. Mitchell and Sebastian Thrun. “Explanation based learning: A comparison of symbolic and neural network approaches”. In Paul E. Utgoff, editor, Proceedings of the Tenth International Conference on Machine Learning, pages 197–204, SanMateo, CA, 1993.MorganKaufmann.
8. Baxter J., Tridgell A., Weaver L., “KnightCap: A chess program that learns by combining TD(lambda) with gametree search”, Proceedings of the Fifteenth International Conference on Machine Learning,, July (1998), pp. 28-36.
9. Beal D. F., Smith M. C., “Learning Piece values using Temporal Difference”, Journal of the International Chess Association, September (1997).
10. Bowden B. V., “Faster Than Thought”, Chapter 25, Pitman, (1953).