# A Trappy Alpha-Beta Search Algorithm in Chinese Chess Computer Game

Jian Fang, Jian Chi, Hong-Yi Jian
Mathematics and Computer Department, Hebei Normal University for Nationalities, Chengde 067000, Hebei, China
Email: csjfang@yeah.net

***Abstract*—In this paper, we propose an improved alpha-beta search algorithm, named trappy alpha-beta (simply $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$), for game-tree in order to identify and set the potential traps in the game playing. $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ can be regarded as an extension of the traditional alpha-beta search algorithm which ties to predict when the opponent might make a mistake and select such moves that can most likely lead the an opponent into the trap by comparing the various scores returned through iterative deepening technology. In $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$, we define two basic components: 1) defining a trap by considering the nature of alpha-beta search algorithm and referring the evaluation value returned by iterative deepening; and 2) evaluating a trap by calculating the probability that the opponent fall into the trap and the advantage followed when the opponent fall into it. In our experiment, we test the performance of $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ in comparison with three game-tree search algorithms, i.e., min-max, trappy minimax, and alpha-beta, by playing with four testing opponents (their depthes are 7, 8, 9, and 10 respectively), which are obtained form a typical Chinese chess computer game programme-Xqwizard (*http://www.xqbase.com/*). The comparative results show that our designed $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ can effectively find and set the traps in the playing with opponents.***

***Keywords*—alpha-beta search; Chinese chess computer game; game-tree; iterative deepening; min-max search; trappy minimax**

## I. Introduction

Shannon in 1950 [1] firstly propose how to design a chess-playing programme which should include three major components: move generation [2], evaluation function [3], and move search algorithm [4]. By analyzing the 3-ply game-tree in Fig. 1, we give the simple descriptions about these three parts as follows:

1) The move generation [2] is represented as a game-tree which organizes the moves generated in the playing process with a tree structure as shown in Fig. 1. The root node of game-tree represents the current playing position and its children nodes are the subsequent positions that are generated by carrying out all the feasible moves. Every children node continues to extend their subsequent positions according to the above-mentioned process until the specific depth is arrived.
2) However, in the practical implementation, due to the limitations of running time and memory requirement, the game-tree can not extend to such positions in which the win or failure is clear. Thus, we need to assess the positions (e.g., leaf nodes in Fig. 1) with a evaluation function [3] by extracting some features from the position, such as material balance, adjunctive value of position, mobility, board control and connectivity. Through assigning weights to each feature, the evaluation function is able to convert a position into a score (e.g., the digits under leaf nodes in Fig. 1).
3) The search algorithm [4] is used to find an best move for the root node in the game-tree by comparing the different returned values from the leaf nodes. The commonly used game-tree search algorithms are min-max [5] and alpha-beta [6]. Alpha-beta can be regarded as a pruned min-max search algorithm, because when searching the game-tree, min-max needs to construct a total game-tree, while alpha-beta establishes a pruned game-tree. The size of game-tree generated by alpha-beta is always smaller than alpha-beta.

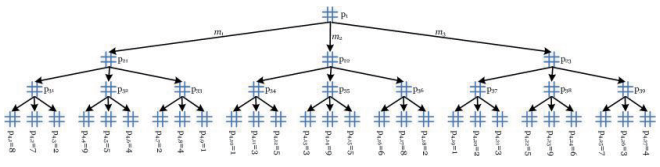

Figure 1. An illustration of game-tree

For the move generation and evaluation function, in recent years there are many representative works [7, 8, 9, 10] which have been proposed and obtained the successful applications in Backgammon, Go, Checker, Othello, International chess, and Chinese chess, etc, two-player board games. For example, Fenner and Levene [7] used the hashing scheme to design the bit-board move generation for moving the pieces in chess-like board games. In their development, the rotated bit-boards are unnecessarily considered and the finally experimental results show that the simple variations of hashing functions will bring about a minimal perfect hashing scheme. By studying the brain activity of professional and amateur players in a board game named shogi with the functional magnetic resonance imaging, Wan, er al, [8] found that there are two specific activations which can influence the professionals in the game-playing, i.e., one is the precuneus of the parietal lobe during perception of board patterns, and the other is the caudate nucleus of the basal ganglia during quick generation of the best next move. Based on the individual evaluations according to played games through the several generations and under the different environments, the authors in [9] presented a differ-

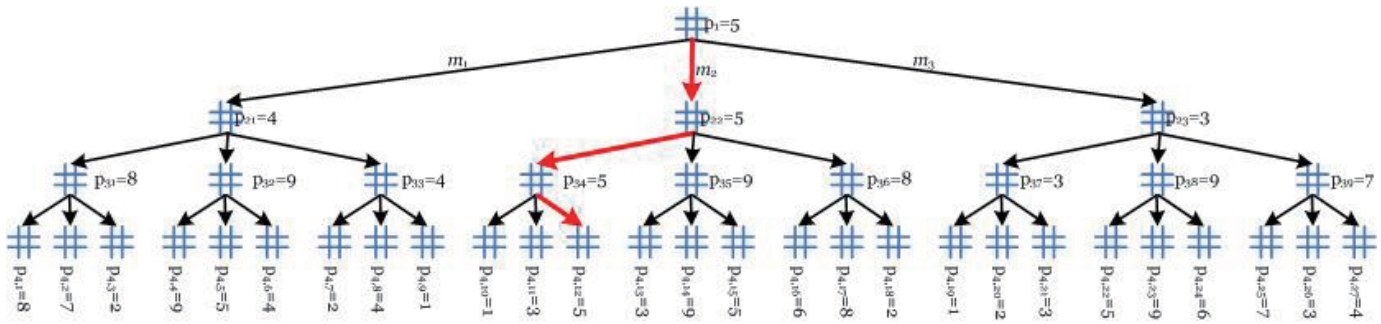


Figure 2. The min-max search algorithm

ential evolution (simply DiffE) algorithm to evaluate and tune the position evaluation function. By employing and upgrading with a history mechanism, DiffE used an opposition-based optimization to improve the evaluation of individuals and the tuning process. Vazquez-Fernandez, et al, [10] used a local search scheme based on the Hooke-Jeeves algorithm to construct the evaluation function, which is adopted to adjust the weights of the best virtual player obtained in the evolutionary process. In this paper, our study mainly focuses on designing a high-intelligent search algorithm by introducing the trappy mechanism [11]. Gordon and Reda [11] firstly proposed a trappy minimax search algorithm, which is called $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$ in this paper, in *2006 IEEE Symposium on Computational Intelligence and Games*. Motivated by $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$, we propose an improved alpha-beta search algorithm, named trappy alpha-beta (simply $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$), for game-tree in order to identify and set the potential traps in the game playing. $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ can be regarded as an extension of the traditional alpha-beta search algorithm which ties to predict when the opponent might make a mistake and select such moves that can most likely lead the an opponent into the trap by comparing the various scores returned through iterative deepening technology. In the experimental part, we test the performance of $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ in comparison with three game-tree search algorithms, i.e., min-max, $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$, and alpha-beta, by playing with four testing opponents (their depthes are 7, 8, 9, and 10 respectively), which are obtained form a typical Chinese chess computer game programme-Xqwizard (*http://www.xqbase.com/*). The comparative results show that our designed $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ can effectively find and set the traps in the playing with opponents.

## II. The Brief Descriptions of Three Game-Tree Search Algorithms

### A. Min-max Search

According to Fig. 2, we explain the running process of min-max search algorithm. In Fig. 2, the nodes in odd plies are called max-nodes, e.g., $\mathrm{p}_1$, $\mathrm{p}_{31}$, $\mathrm{p}_{32}$, $\cdots$, $\mathrm{p}_{39}$. And, the nodes in even ply are called min-nodes, e.g., $\mathrm{p}_{21}$, $\mathrm{p}_{22}$, $\mathrm{p}_{23}$, $\mathrm{p}_{41}$, $\mathrm{p}_{42}$, $\cdots$, $\mathrm{p}_{4,27}$. The values of max-nodes are the maximal values of their children nodes, e.g.,

$$\mathrm{p}_1 = \max\left(\mathrm{p}_{21}, \mathrm{p}_{22}, \mathrm{p}_{23}\right) = \max\left(4, 5, 3\right) = 5, \text{ and} \tag{1}$$

$$\mathrm{p}_{34} = \max\left(\mathrm{p}_{4,10}, \mathrm{p}_{4,11}, \mathrm{p}_{4,12}\right) = \max\left(1, 3, 5\right) = 5. \tag{2}$$

And, the values of min-nodes are the minimal values of their children nodes, e.g.,

$$\mathrm{p}_{22} = \min\left(\mathrm{p}_{34}, \mathrm{p}_{35}, \mathrm{p}_{36}\right) = \min\left(5, 9, 8\right) = 5, \text{ and} \tag{3}$$

$$\mathrm{p}_{23} = \min\left(\mathrm{p}_{37}, \mathrm{p}_{38}, \mathrm{p}_{39}\right) = \min\left(3, 9, 7\right) = 3. \tag{4}$$

According to the principle mentioned above, the position A can find the best move $m_5$ which will bring about the maximum advantage for current player and meanwhile give rise to the maximum obstacle for the opponent.

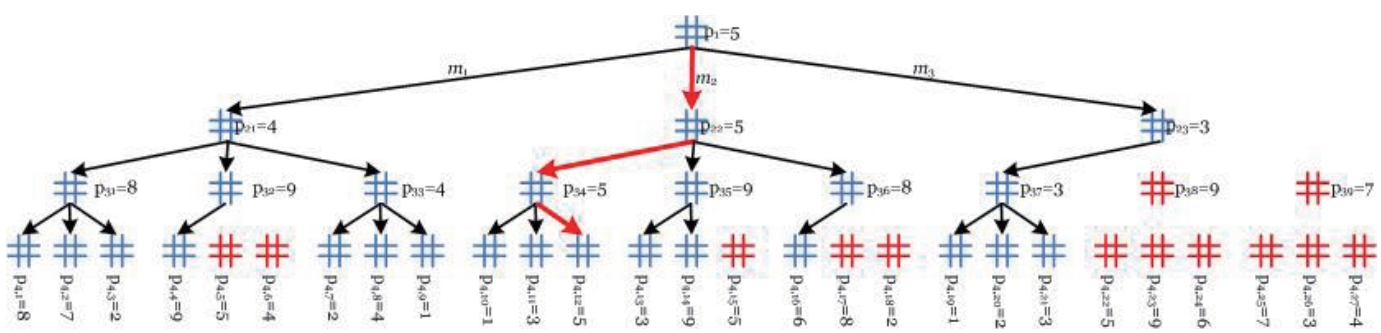


Figure 3. The alpha-beta search algorithm

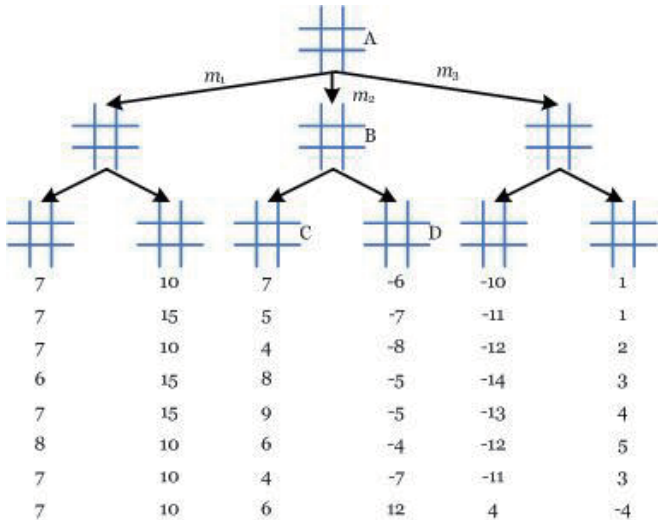


Figure 4. $\mathrm{Trap}^{\mathrm{Min}}_{\mathrm{Max}}$ [11]

### B. Alpha-beta Search

Alpha-beta search can be regarded as a pruned min-max search, because in the process of game-tree searching, some nodes are not necessary to generate. This will save a great deal of running time and memory requirement. Fig. 3 gives the game-tree generated with alpha-beta search. By observing Fig. 3, we depict the running process of alpha-beta search.

**Alpha-pruning:** In Fig. 3, the value of node $\mathrm{p}_1$ can be calculated with the following equation:

$$\begin{aligned}\mathrm{p}_1 &= \max(\mathrm{p}_{22}, \mathrm{p}_{23}) = \max(\mathrm{p}_{22}, \min(\mathrm{p}_{37}, \mathrm{p}_{38}, \mathrm{p}_{39})) \\ &= \max(5, \min(3, \mathrm{p}_{38}, \mathrm{p}_{39})) = 5.\end{aligned} \tag{5}$$

From Eq. (5), we can see that the values of nodes $\mathrm{p}_{38}$ and $\mathrm{p}_{39}$ do not have an impact on the calculation of $\mathrm{p}_1$. So, in the process of searching game tree, we can give up the generation to the nodes $\mathrm{p}_{38}$ and $\mathrm{p}_{39}$. This search strategy is called **alpha-pruning**.

**Beta-pruning:** In Fig. 3, the value of node $\mathrm{p}_{21}$ can be calculated with the following equation:

$$\begin{aligned}\mathrm{p}_{21} &= \min(\mathrm{p}_{31}, \mathrm{p}_{32}) = \min(\mathrm{p}_{31}, \max(\mathrm{p}_{44}, \mathrm{p}_{45}, \mathrm{p}_{46})) \\ &= \min(8, \max(9, \mathrm{p}_{45}, \mathrm{p}_{46})) = 8.\end{aligned} \tag{6}$$

From Eq. (6), we can see that the values of nodes $\mathrm{p}_{45}$ and $\mathrm{p}_{46}$ do not have an impact on the calculation of $\mathrm{p}_{21}$. So, in the process of searching game tree, we can give up the generation to the nodes $\mathrm{p}_{45}$ and $\mathrm{p}_{46}$. This search strategy is called **beta-pruning**.

### C. $\mathrm{Trap}^{\mathrm{Min}}_{\mathrm{Max}}$

Through the example in Fig. 4, we introduce the basic idea of $\mathrm{Trap}^{\mathrm{Min}}_{\mathrm{Max}}$. In Fig. 4, the computer has three moves from which to choose. Suppose the second move B is chosen. In that case, a score of at least 6 is guaranteed, presuming the opponent plays the best response C. However, the alternative for the opponent D looks very appealing when evaluated at depths 3 through 9. However, when evaluated at depth 10, the node has an evaluation that is considerably worse than even the correct move, despite all the good evaluations using the upper levels.

**Algorithm 1** Trappy Alpha-Beta Search-$\mathrm{Trap}^{\mathrm{A}}_{\mathrm{B}}$

```
Input: The current position p and maximal search depth maxdepth;
Output: The best move of p;
best, rawEval, bestTrapQuality = −∞;
for Every move m corresponding to position p do
    Make move m on position p;
    for Every response of opponent do
        scores[maxdepth] = - alpha-beta(p, maxdepth);
        if scores[maxdepth] > rawEval then
            rawEval = scores[maxdepth];
        end if
    end for
    Tfactor = Trappiness(scores[]);
    profit = scores[maxdepth]-rawEval;
    trapQuality = profit × Tfactor;
    if trapQuality > bestTrapQuality then
        bestTrapQuality = trapQuality
    end if
    adjEval = rawEval + scale(bestTrapQuality)
    if adjEval > best then
        best = adjEval;
    end ifRetract move m from position p;
end for
Return best;
```

## III. Trappy Alpha-Beta Search-$\mathrm{Trap}^{\mathrm{A}}_{\mathrm{B}}$

In Algorithm 1, we give the description of $\mathrm{Trap}^{\mathrm{A}}_{\mathrm{B}}$. In Algorithm 1, two necessary parameters are needed to determine: trappiness, and profit (i.e., profitability).

1) **Trappiness** is based on the distance between a high positive score and an actual negative score, and is calculated from the point of view of the opponent. That is, it attempts to measure the likelihood that the opponent could miss the trap.
2) **Profitability** is the gain to the program if the opponent falls for the trap. It is calculated from the point of view of the algorithm.

Trappiness and profitability are both factored into the evaluation of each possible computer move, along with the alpha-beta search as shown in subsection II-B. The trappiness factor (T) has a range from $[0, 1]$, of which the calculations are different depending on the backed-up score for alpha-pruning level (i.e., odd ply) or beta-pruning level (i.e., even ply). T corresponding to the alpha-pruning level is defined as:

$$\mathrm{T} = \begin{cases} 0 & u \le m \\ 0.75\,\frac{u-m}{abs(m)} & m < u < m + abs\,(m) \\ 0.75 + 0.25\,\frac{u-m-abs(m)}{3abs(m)} & m + abs\,(m) \le u < m + 4abs\,(m) \\ 1 & u \ge m + 4abs\,(m) \end{cases}$$

And, $\mathrm{T}$ corresponding to the beta-pruning level is defined as:

$$\mathrm{T} = \begin{cases} 0 & u \geq m \\ 0.75\,\frac{m-u}{abs(m)} & m > u > m - abs\,(m) \\ 0.75 + 0.25\,\frac{m-u-abs(m)}{3abs(m)} & m - abs\,(m) \geq u > m - 4abs\,(m) \\ 1 & u \leq m - 4abs\,(m) \end{cases}$$

where, $u$ is the evaluation value of position at the $\mathrm{maxdepth}-1$ ply and $m$ is the evaluation value at the $\mathrm{maxdepth}$ ply.

In our $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$, we define a **trap** as follows: A trap is a move that looks good in the short term but has bad consequences in the long term. Thus, in the alpha-beta algorithm, it is a move with high evaluations at shallow depths and a low evaluation at the $\mathrm{maxdepth}$ ply. Traps have the significance that a non-optimal opponent might be tricked into thinking that they are good moves when in fact they are not.

## IV. Experimental Setup and Analysis

In our experiment, we test the performance of $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ in comparison with three game-tree search algorithms, i.e., min-max, $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$, and alpha-beta, by playing with four testing opponents (their depthes are 7, 8, 9, and 10 respectively), which are obtained form a typical Chinese chess computer game programme-Xqwizard (*http://www.xqbase.com/*).

Our comparisons are arranged as following procedures: Firstly, let $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ with 7, 8, 9, and 10 depths play with min-max, $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$, and alpha-beta with the same search depths respectively. Secondly, for every search depth, $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ plays with its opponents 100, 200, 300, 400, and 500 times respectively. Thirdly, the playing results, i.e., the numbers of win and lose, are recorded. The experimental results are listed in TABLE I.

From TABLE I, we can see that because without using any trappy strategy, $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ obtains the significantly better performances compared with min-max and alpha-beta. For example, compared with min-max with 7, 8, 9, and 10 depths, the winning percentages of $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ arrive 0.867, 0.837, 0.863, and 0.850 respectively. And, compared with alpha-beta with 7, 8, 9, and 10 depths, the winning percentages of $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ arrive 0.857, 0.856, 0.842, and 0.833 respectively. In comparison with $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$, our $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ also obtains the better performances, i.e., the average winning percentage can also reach 80%. Because in the process of generation of game-tree, alpha-beta can prune more necessary nodes, it can search more deeply within the same time and memory limitations. Hence, $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ can find more hidden traps than $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$. The comparative results show that our designed $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ can effectively find and set the traps in the playing with opponents. The results also reflect the opponents that performs a full-width search to a depth greater than $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ will not fall for a trap. However, when the opponents do not perform full-width search, they are always susceptible to traps set by $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$.

TABLE I
THE EXPERIMENTAL RESULTS BY PLAYING WITH DIFFERENT TESTING PLAYERS

| $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ | vs. Min-max | vs. $\mathrm{Trap}_{\mathrm{Max}}^{\mathrm{Min}}$ | vs. Alpha-beta |
|---|---|---|---|
| 7-ply | 80-20 | 73-27 | 68-32 |
| 7-ply | 160-40 | 142-58 | 124-76 |
| 7-ply | 246-54 | 228-72 | 188-112 |
| 7-ply | 328-72 | 280-120 | 270-130 |
| 7-ply | 422-78 | 397-103 | 323-177 |
| 8-ply | 84-16 | 79-21 | 65-35 |
| 8-ply | 164-36 | 154-46 | 137-63 |
| 8-ply | 240-60 | 231-69 | 191-109 |
| 8-ply | 354-46 | 300-100 | 269-131 |
| 8-ply | 421-79 | 365-135 | 309-191 |
| 9-ply | 82-18 | 77-23 | 63-37 |
| 9-ply | 171-29 | 143-57 | 134-66 |
| 9-ply | 251-49 | 236-64 | 206-94 |
| 9-ply | 344-56 | 300-100 | 276-124 |
| 9-ply | 441-59 | 382-118 | 341-159 |
| 10-ply | 87-13 | 73-27 | 63-37 |
| 10-ply | 167-33 | 151-49 | 135-65 |
| 10-ply | 249-51 | 235-65 | 197-103 |
| 10-ply | 335-65 | 308-92 | 262-138 |
| 10-ply | 422-78 | 385-115 | 331-169 |

## V. Conclusion

In this paper, we propose an improved alpha-beta search algorithm, named trappy alpha-beta (simply $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$), for game-tree in order to identify and set the potential traps in the game playing. In $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$, we define two basic components: how to define and evaluate a trap by calculating the probability that the opponent fall into the trap. The finally comparative results show that our designed $\mathrm{Trap}_{\mathrm{B}}^{\mathrm{A}}$ can effectively find and set the traps in the playing with opponents.

## References


[1] C. E. Shannon, "Programming a computer for playing chess," Philosophical Magazine, vol. 41, no. 314, pp. 256-275, 1950.

[2] J. J. Gillogly, "The technology chess program," Artificial Intelligence, vol. 3, pp. 145-163, 1972.

[3] J. Clune, "Heuristic evaluation functions for general game playing," In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1134-1139, 2007.

[4] Michael Tarsi, "Optimal search on some game trees," Journal of the ACM, vol. 30, no. 3, pp. 389-396, 1983.

[5] M. S. Campbell, T. A. Marsland, "A comparison of minimax tree search algorithms," Artificial Intelligence, vol. 20, no. 4, pp. 347-367, 1983.

[6] J. Schaeffer, "The history heuristic and alpha-beta search enhancements in practice ," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 11, pp. 1203-1212, 1989.

[7] T. Fenner, M. Levene, "Move generation with perfect hash functions," International Computer Games Association Journal, vol. 31, no. 3, pp. 3-12, 2008.

[8] X. H. Wan, H. Nakatani, K. Ueno, T. Asamizuya, K. Cheng, K. Tanaka, "The neural basis of intuitive best next-move generation in board game experts," Science, vol. 331, no. 6015, pp. 341-346, 2011.

[9] B. Boskovic, J. Brest, A. Zamuda, S. Greiner, V. Zumer, "History mechanism supported differential evolution for chess evaluation function tuning," Soft Computing-A Fusion of Foundations, Methodologies and Applications, vol. 15, no. 4, pp. 667-683, 2010.

[10] E. Vazquez-Fernandez, C. A. C. Coello, F. D. S. Troncoso, "An evolutionary algorithm coupled with the Hooke-Jeeves algorithm for tuning a chess evaluation function," In Proceedings of 2012 IEEE Congress on Evolutionary Computation, pp. 1-8, 2012.

[11] V. S. Gordon, Ahmed Reda, "Trappy minimax-using iterative deepening to identify and set traps in two-player games," In Proceedings of 2006 IEEE Symposium on Computational Intelligence and Games, pp. 205-210, 2006.