EasyChair Preprint

№ 12767

Discovery of the Characteristics of the Cubic Othello Chessboard and Its Implementation of Visualization Expert System

Jer Fong Chen and Fang Rong Hsu

March 27, 2024

Discovery of the Characteristics of the Cubic Othello Chessboard and its Implementation of Visualization Expert System

JER-FONG CHEN, AND FANG-RONG HSU

Department of Information Engineering and Computer Science, Feng Chia University, Taichung 407, Taiwan

Abstract

The standard planar Othello game has 8 (rows) * 8 (columns) = 64 (squares). Although this planar Othello game has been solved, this paper still proposes a new problem based on this foundation: Has the originally evenly matched situation between the first move and second move sides changed in the game of Othello played on a cubic board?

Certainly, we can start constructing a cubic Othello board from 8 (rows) * 8 (columns) * 6 (faces) = 384 (squares), but a board with 384 (squares) is not suitable for people to play against each other. Even if it is changed to the smallest cubic Othello board with 4 (squares) * 6 (faces) = 24 (squares), it is not very suitable for people to play against each other, because playing cubic Othello in a real-world environment is very difficult. Playing cubic Othello is very challenging because it requires understanding the characteristics of cubic Othello and making clear definitions of the ways of moving. Only in this way can we attempt to apply machine learning techniques to cubic Othello and develop an expert system that can play chess on a cubic Othello board.

Then we found that the first player (black) of cubic Othello has a winning rate of about 20% with blind moves, while the second player (white) has a probability of about 80% with blind moves. It is evident that in cubic Othello, the second player has an advantage, which is far from the conclusion pointed out in the paper related to planar Othello, that a perfect game of two players in planar Othello will eventually lead to a draw.

Furthermore, the expert system proposed in this paper that is trained using the GPU memory on a personal computer can be executed on any contemporary browser. The training that originally took tens of days to complete using CPU memory on a personal computer can now be completed in tens of minutes in the GPU memory of a personal computer. This clearly shows that the significant benefits that can be achieved by effectively utilizing the GPU memory on a personal computer have surpassed the use of a large CPU memory computer.

Index Terms: Othello Game, Cubic Chessboard, Artificial Intelligence, Machine Learning, K-Nearest Neighbors Algorithm, GPU Memory

1. Introduction

The earliest self-play in planar Othello began with Logistello[1]. In August 1997, Logistello defeated the 1996 Othello world champion Takeshi Murakami. Subsequently, various classical board games have been conquered by computer AI. It is evident that in virtual battles, whether there is computer assistance has become the key to the outcome of the battle.

The rules for placing pieces in cubic Othello and planar Othello are the same. After placing a piece, if any of your own pieces on the board are on the same line (horizontal, vertical, or diagonal) and there are opponent's pieces in between, you can flip these sandwiched opponent's pieces to your own (just flip them over). When neither side can place a piece according to the above conditions, the game ends and the player with more pieces of their color wins. Therefore, to determine whether an empty space on the cubic Othello board can be placed, the cube must be spread out into a plane.

The difference in difficulty between cubic (4 * 6) Othello and planar (8 * 8) Othello can be shown in Figure 1. Excluding the first move, cubic (4 * 6) Othello has approximately $3.53 * 10^{12}$ routes, while planar (8 * 8) Othello has approximately $5.77 * 10^{52}$ routes.

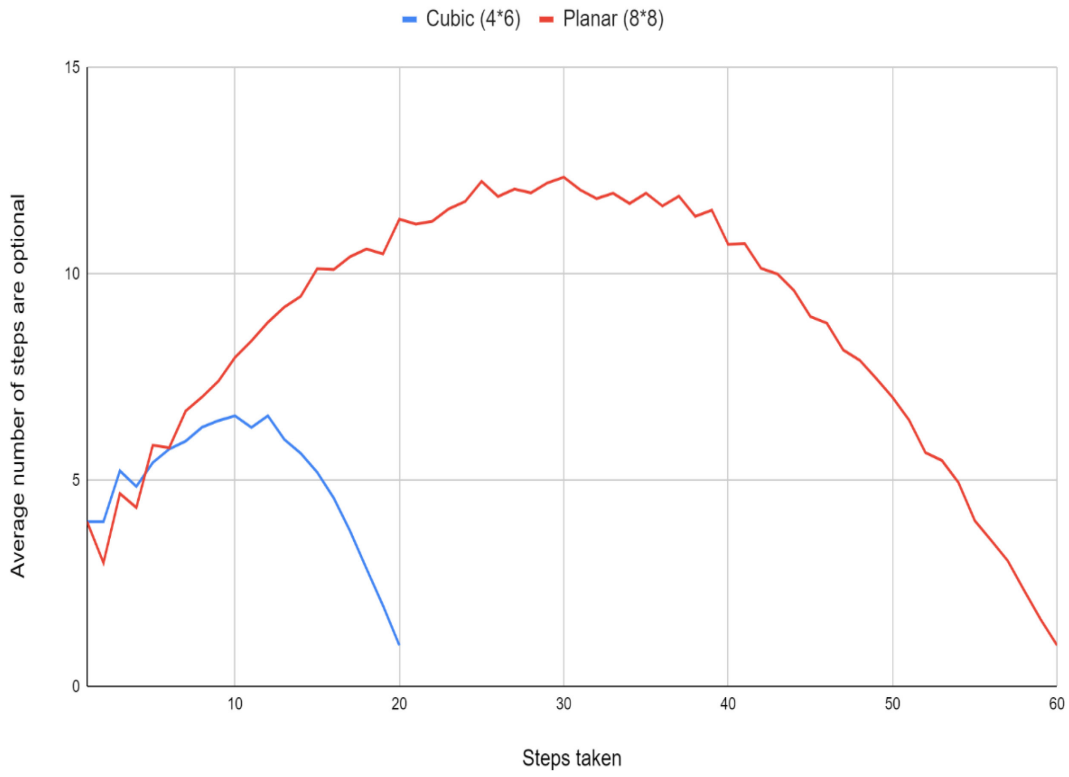

Figure 1. Comparison of the average number of positions that can be chosen at each step between cubic Othello and planar Othello.

However, real battles do not exist in purely planar battlefields, but more often in cube space battlefields. Therefore, there is a necessity for system simulation of cubic Othello chessboards. In the early days, when GPU memory was not widely applied, executing machine learning models

could take more than a week. Thus, this paper also takes the opportunity to demonstrate how to utilize GPU memory on personal computers or smartphones to quickly perform machine learning computations in the development of this cubic and planar Othello expert system.

2. The Characteristics of the Cubic Othello Chessboard

In a cubic chessboard, each square has 7 adjacent squares (including diagonally), unlike a planar chessboard where there are corners with only 3 adjacent squares, and edges with 5 adjacent squares. This leads to a strategy in a planar chessboard where players or machines tend to take corners and avoid losing corners. Moreover, although each square on a cubic chessboard only has 7 adjacent squares (a regular square on a planar has 8 adjacent squares), the squares on a cubic chessboard have five attack loops (a loop is a line that passes through the square when the cube is flattened in a way that each face is connected). This is one more than the lines that a regular square on a planar chessboard can attack (as shown in Figure 2).

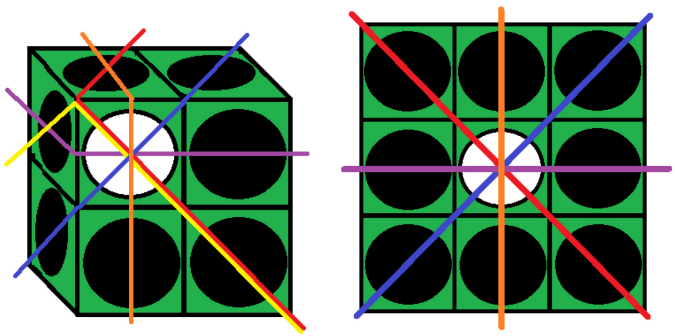

Figure 2. Display the directional lines that the white piece can attack on the cubic chessboard and the planar chessboard.

In addition, taking the position of the white pieces on the cubic chessboard in Figure 2, marked as 30, as an example, the various colored loop lines are spread out on the plane after the cube is unfolded, as shown in Figures 3 to 7.

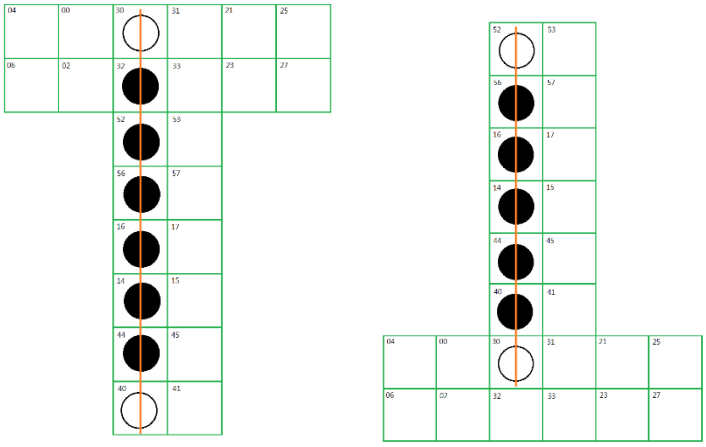

Figure 3. The loop line surrounding the left face of the cube.

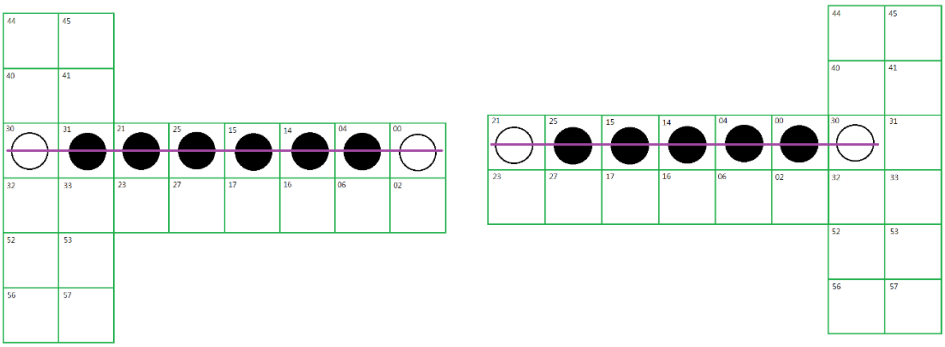

Figure 4. The loop line surrounding the top face of the cube.

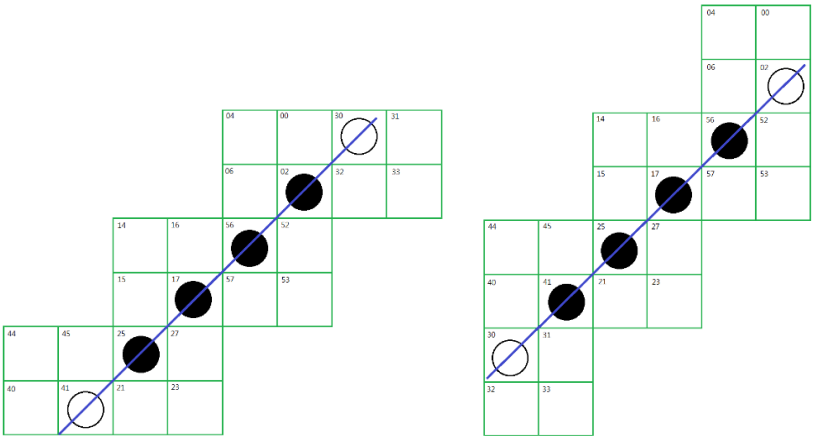

Figure 5. The axis around which the loop line is wrapped is composed of the point formed by the three faces of 04, 14, 44 and the point formed by the three faces of 53, 23, 33.

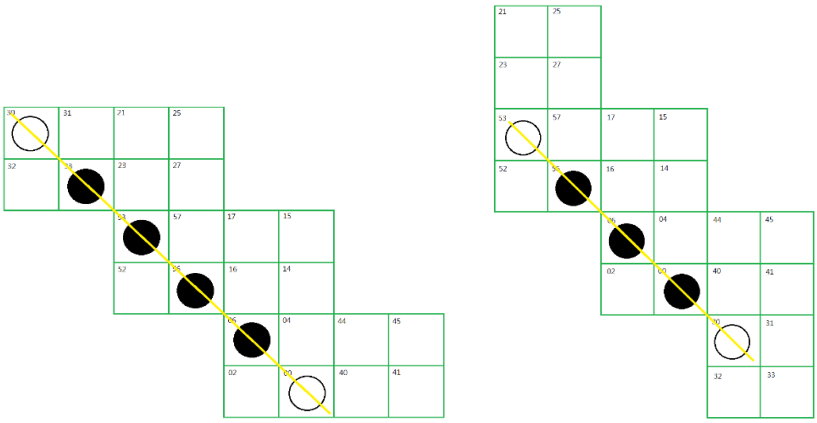

Figure 6. The loop lines surround ths point formed by the three faces of 32, 52, 02.

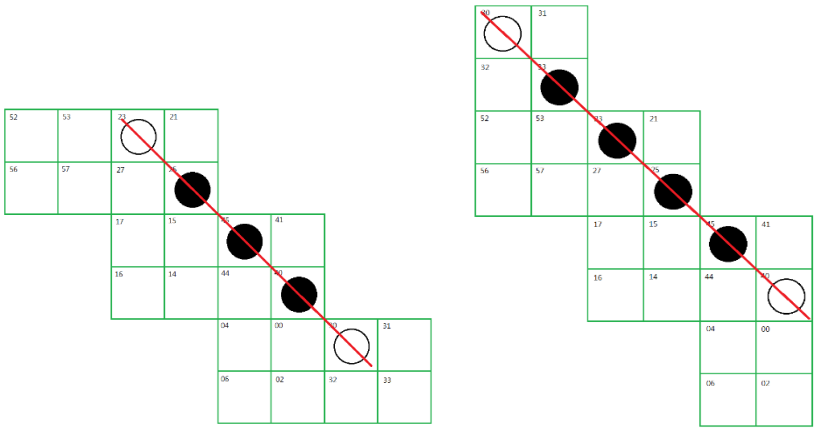

Figure 7. The loop lines surround ths point formed by the three faces of 21, 41, 31.

At this point, a new question arises: for any position on the cubic chessboard, besides the aforementioned 5 types of attack lines, are there any other attack lines? The answer is yes, after flattening the cubic board, there are other straight lines that can be considered as attack lines. However, only these 5 types of attack lines are not only loop lines, but also the faces of the cube are connected by lines instead of only a point after the cube is flattened. In addition, after placing a piece, how to obtain the attack lines that can flip the opponent's pieces, as shown in Algorithm 1.

Algorithm 1. *CL*(*p*, *h*, *c*): Find the attack lines that can flip the chess pieces into mine.

Require: *p*: The position object *p* of the square to place the piece, this position object must contain an array of attack lines, and each line in the array of attack lines must record the coordinates of the positions passed by the attack line.
Require: *h*: The current situation array *h* of the chessboard, the array will record the current situation of each position coordinate.
Require: *c*: The situation *c* of the current placing piece.
Ensure: *h* only recorded the following three situations: black piece, white piece, and no piece. The three situations are mutually exclusive and do not overlap.

```
o ← {}
for l ∈ p.lines do
    f ← 0
    t ← h[l[f + 1].coordinate]
    while t is not c and t is not no piece do
        f ← f + 1                                  ▷ f is not bigger then l.length
        t ← h[l[f + 1].coordinate]
        if t is c then
            o ← o + l, f
            break while
        end if
    end while
end for
return o
```

From Algorithm 1, assuming n is the number of attack lines at a certain position, and m is the number of squares that this attack line passes through, then the time complexity of the attack line increasing is O(n), and the space complexity is O(n × m), so the increase of the chessboard size cause insufficient memory, the increase of the attack line at each position will also cause

insufficient memory. Therefore, choosing meaningful loop lines as attack lines can reduce the difficulty of implementing the expert systems of the cubic Othello.

In addition, playing 100 rounds of random games of Othello on a plane and a cube, it can be observed that on the cubic Othello, the player with the second move (white pieces) has an advantage, as shown in Table 1.[2]

Table 1. Compare whether the first move has an advantage or the second move has an advantage in the plane and the cube.

Playing 100 rounds of random games	Black Win (first move)	White Win (second move)	Tie
Planar Othello	46	48	6
Cubic Othello	16	**82**	2

3. Method

3.1 Backtracking[3]

After trying Q-learning[4] (Q table) and multithreading, it was found that the fastest way to obtain battle records is still to execute single-threaded backtracking. However, the method of recording wins and losses adopts a reward method similar to reinforcement learning[5]. The goal is to find if there is a path for the black piece (first move) to win, and adopt a black-winner-takes-all strategy. Let each square on the path to victory for the black piece have a reward value of -24 (if there is only one black piece on the board, it is -1, up to 24, so it is -24). On the contrary, if this game has ended and the white piece still wins, then the squares on this path will maintain their original reward value (the original reward value refers to the reward value at the position after a piece is placed, which is the number of white pieces on the board minus the number of black pieces, and all possible next steps The reward value is set to -25, indicating that any reward value generated in the next step can replace it).

3.2 K-nearest neighbor algorithm[6]

After obtaining the battle record, the next step is to train the computer to establish a model. After comparing neural networks (NN) and K-nearest neighbor algorithm (KNN), it was found that neural networks have difficulty obtaining feature values between square reward values and winning paths. The reason is that after each move in Othello, the change in reward value is too large, causing training errors to be difficult to converge. Therefore, we adopt the K-nearest neighbor algorithm that requires a large amount of calculation. Fortunately, now is the era of GPU memory. Since Google opened TensorFlow.js[7], we have entered an era where large-scale calculations can be performed on the front end of web pages. In addition, ml5.js[8] makes TensorFlow.js more friendly to programmers. After that, machine learning technology has become easier to apply in various fields.

3.3 Prediction

The predictive data of KNN is based on the data obtained from the backtracking method. The backtracking data used in this paper for cubic Othello is a path where there are no empty squares on the board when the game ends. Then, all path data within 9 steps of backtracking from this path is collected. For the 8 * 8 planar chessboard of our paper (used as a control group), applying the above logic can only backtrack the path data within 7 steps. This is done to ensure that the usage of GPU memory does not exceed 8GB.

The principle of black pieces (first move) is to try not to let white pieces go to positions where the reward value is not -24. On the contrary, the principle of white pieces is to try to find positions where the reward value is not -24. As long as a white piece goes to a position where the reward value is not -24, it means that white pieces are almost certain to win, except for errors in KNN prediction, black pieces may win.

Comparing with the paper we referred to, "Othello is Solved"[2], that first finds all the results of the first ten steps (where there are still 50 squares left to play) on an 8 * 8 planar chessboard, and then uses α-β search to remove unwanted paths from the situation presented by that game, until there are only 36 squares left to play. Finally, it calculates the game-theoretic value of all 36 squares that meaning all possible paths for the remaining 36 squares are run, and the optimal path leading to a tie. However, this referenced paper cannot explain the phenomenon caused by the prediction of the expert system in this paper.

4. Implementation

The cubic Othello and the planar Othello implementation URL is: https://www.cpapeijer.com/home/autolearn2. It is written in JavaScript and uses the public packages p5.js[9] and ml5.js. The model trained under the condition of not exceeding 8GB GPU memory can be smoothly executed on any computer or mobile phone that can run a browser for prediction.

5. Results and Discussion

Through GPU memory computation, the workload that originally required several days to complete with CPU memory can be completed in just a few minutes, and the time spent on predicting answers can also be completed in a few seconds.

In the implementation of the expert system for cubic Othello, we not only compared battles in expert vs. random and expert vs. expert, but also implemented an expert system for planar Othello as a control group because the results on cubic Othello were too strange. The experimental results are shown in Table 2 and Table 3.

Table 2. Comparison of playing with and without the expert system. (cubic version)

Playing 100 rounds of cubic Othello games	Black Win (first move)	White Win (second move)	Tie
Black Expert vs. White Expert	0	100	0
Black Random vs. White Expert	15	82	3
Black Expert vs. White Random	31	67	2

Table 3. Comparison of playing with and without the expert system. (planar version)

Playing 100 rounds of planar Othello games (control group, GPU memory out of 8GB)	Black Win (first move)	White Win (second move)	Tie
Black Expert vs. White Expert	38	62	0
Black Random vs. White Expert	37	60	3
Black Expert vs. White Random	57	42	1

Referring to the results in Table 1, it is evident that using the expert system implemented with KNN improves the winning rate by 10% (the opponent's win rate decreases by 10%) under the condition of using limited GPU memory, compared to the method of randomly placing pieces without using the expert system, resulting in an overall difference of almost 20%. However, when both sides use the expert system, the white pieces (second move) have an advantage, especially noticeable in the cubic Othello game. This may be attributed to using the backtracking method, the path data used to train KNN is set to an over-optimistic belief in victory after the black pieces win, but it allows the white pieces to find the winning opportunity that is like a light in the dark.

6. Conclusion

In this era where artificial intelligence triumphs over human chess masters, this paper advances the world of chess gaming into the cubic chessboard, and discovers some astonishing characteristics in it, such as the number and selection of attack lines, the advantages of going first or second, and the reaction of artificial intelligence to the situation of asymmetric advantages. These are new discoveries made in the process of implementation. Of course, in the future, more flat chess games may be cubed, because there are always some people who are not afraid to try that is too difficult.

Certainly, we also recognize that some readers may be skeptical about the effectiveness of the expert system proposed in this paper. But in fact, the focus of this paper is on the application of GPU memory, not on the development of new artificial intelligence or machine learning technologies. At present, there are fewer papers discussing the use of front-end GPU memory on web pages. Although HTML 5 already included the WebGL[10] 3D drawing framework that uses GPU memory since 2006, it wasn't until the emergence of TensorFlow.js in 2018 that the threshold for using WebGL to build artificial intelligence in the browser was lowered. The ml5.js announced in the same year is a user-friendly version of TensorFlow.js. So far, the trilogy of machine learning (collection, training, and prediction) can be more easily executed on the front end of the

browser. This paper is just a starting point, hoping that in the future there will be more artificial intelligence applications on the front end of the browser, so that the construction of artificial intelligence is not exclusive to certain professionals, but like cooking, as long as you want to do it for the people you love, everyone can become a master.

7. Acknowledgments

The author acknowledges that translations and some code examples for this article were generated by ChatGPT (powered by OpenAI's language model; http://openai.com). The editing was performed by the author.

References

[1] M. Buro (1995), *Logistello: A Strong Learning Othello Program*, 19th Annual Conference Gesellschaft für Klassifikation e.V., Basel

[2] Hiroki Takizawa (November 15, 2023), *Othello is Solved*, arXiv (2023). DOI: 10.48550/arxiv.2310.19387

[3] Gurari, Eitan (1999), "CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms". Archived from the original on 17 March 2007

[4] Watkins, C.J.C.H. (1989), *Learning from Delayed Rewards*, PhD Thesis, University of Cambridge, England

[5] A. L. Samuel (1959), "Some Studies in Machine Learning Using the Game of Checkers", IBM Journal of Research and Development, vol. 3, no. 3, pp. 210-229

[6] Fix E., Hodges J.L. (1951), "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties", Tech. rep. 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas

[7] Google Research (November 9, 2015), "TensorFlow: Large-scale machine learning on heterogeneous systems", https://tensorflow.org

[8] Daniel Shiffman (July 31, 2018), "A Beginner's Guide to Machine Learning in JavaScript", https://thecodingtrain.com

[9] Lauren McCarthy, Casey Reas, Ben Fry (November 17, 2015), "Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing", Make Community, LLC

[10] Tony Parisi (August 30, 2012), "WebGL: Up and Running: Building 3D Graphics for the Web", O'REILLY