

The Search Algorithm about Reversi

Project 1 Report of
CS303 Artificial intelligence

Department of Computer Science and Engineering

11812109

阮业淳

Fall 2020

Table of Content

1	Preliminaries	3
1.1	Software & Hardware	3
1.2	Algorithms.....	3
1.3	Applications.....	3
2	Methodology	5
2.1	Notation.....	5
2.1.1	Function	5
2.1.2	Chessboard position [1]	5
2.2	Data Structure	6
2.2.1	<i>chessboard</i>	6
2.2.2	<i>candidateList</i>	6
2.2.3	<i>weightBorad</i>	6
2.3	Model design	6
2.3.1	Depth-limited Search Algorithm.....	7
2.3.2	Minimax Search Algorithm	7
2.3.3	Alpha-Beta Pruning Algorithm	7
2.3.4	Iterative Deepening Search Algorithm	8
2.3.5	Design of evaluation function.....	8
2.4	Details of algorithms:.....	9
	Minimax Search Algorithm	9
	Alpha-Beta Pruning Algorithm.....	11
	Iterative Deepening Search Algorithm.....	11
	Evaluation function.....	12
3	Empirical Verification.....	14
3.1	Dataset.....	14
3.2	Performance measure	14
3.3	Hyperparameters.....	14
3.4	Experimental results.....	15
3.5	Conclusion.....	15
	References.....	16

1 Preliminaries

Reversi, also known as othello, anti reveri, is a chess game for two players. Its rules of the game are simple, player by flipping each other's discs, and finally judging the winner by whose discs are more on the chessboard. Usually black discs play first. The two players took turns. As long as the falling disc and any one of current player's discs on the chessboard hold other part discs in a line (horizontal, straight and oblique lines are allowed), these discs of the other party can be transformed into current player. The chessboard size used in this project is 8×8 , which is the most commonly used Reversi chessboard size.

1.1 Software & Hardware

This project is written in Python with editor PyCharm. The main testing platform are Windows 10 home Chinese version (version 10.0.19042.608) with Intel® Core™ i7-7700HQ CPU @ 2.80GHz with 4 cores and 8 threads, and Reversi battle platform provided by the course.

1.2 Algorithms

Similar to other chess games, Reversi can also select the discs to be dropped by estimating the moves of each step on the chessboard. The algorithms used in this report are the Depth-limited Search Algorithm, Minimax Search Algorithm, Alpha-Beta Pruning Algorithm and Iterative Deepening Search Algorithm.

1.3 Applications

The key of this project is the design of algorithm, which requires effective solution in a certain time (such as 5 seconds). The algorithms related to chess have both the same common and different characteristics. Their common feature is to try to search all possible solutions, and the feature of Reversi algorithm is to focus on the position of the chess. The research of Reversi algorithm is helpful to deepen the understanding of search algorithm and related algorithm, and make the machine more intelligent.

In the field of automatic driving, according to the user's needs and the specific situation of urban road, search algorithm and related specific algorithm will find out the most suitable route for car driving. In the field of remote sensing survey and urban modeling, the search algorithm can combine some specific algorithms to find the appropriate flight direction of UAV.

2 Methodology

This section will introduce some notations, data structure, model design and algorithm detail.

2.1 Notation

2.1.1 Function

Evaluate(chessboard, color): The evaluation function of the probability that the player(disc color is *color*) will win for a given *chessboard* and *color*.

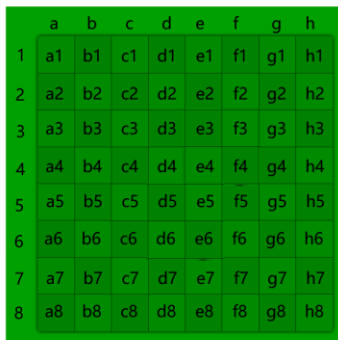
GetNextMoves(chessboard, color): Get all be able to fall disc positions for the player which disc color is *color* for a given *chessboard* and *color*.

GetStableCount(chessboard, color): Get the number of discs that cannot be flipped by the opponent for the player which disc color is *color* for a given *chessboard* and *color*.

2.1.2 Chessboard position [1]

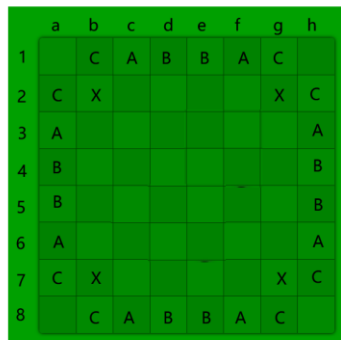
Fig2.1 shows the standard notation for Othello. The columns are labeled 'a' through 'h' from left to right, and the rows are labeled '1' through '8' from top to bottom. Fig2.2 notation was developed by Othello's inventor, Goro Hasegawa, and remains in use today. The B-squares are in the center of the edge, the C-squares are on the edge next to the corner, and the A-squares lie between the B-squares and Csquares. The X-squares are diagonally adjacent to the corners.

At the beginning of the game, the situation of the chessboard is shown in Fig2.3.



	a	b	c	d	e	f	g	h
1	a1	b1	c1	d1	e1	f1	g1	h1
2	a2	b2	c2	d2	e2	f2	g2	h2
3	a3	b3	c3	d3	e3	f3	g3	h3
4	a4	b4	c4	d4	e4	f4	g4	h4
5	a5	b5	c5	d5	e5	f5	g5	h5
6	a6	b6	c6	d6	e6	f6	g6	h6
7	a7	b7	c7	d7	e7	f7	g7	h7
8	a8	b8	c8	d8	e8	f8	g8	h8

Fig2.1



	a	b	c	d	e	f	g	h
1		C	A	B	B	A	C	
2	C	X					X	C
3	A							A
4	B							B
5	B							B
6	A							A
7	C	X					X	C
8		C	A	B	B	A	C	

Fig2.2

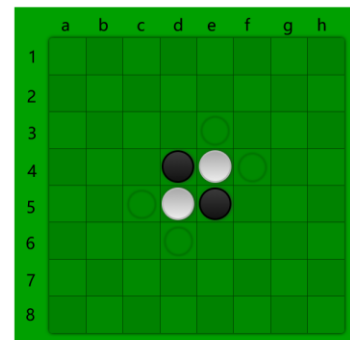


Fig2.3

2.2 Data Structure

2.2.1 *chessboard*

A two-dimensional numpy array, contains the state of disc(none, black, white)

2.2.2 *candidateList*

A list of the most valuable positions which have found.

2.2.3 *weightBorad*

A two-dimensional numpy array, contains the weight value to evaluate the corresponding chessboard position

2.3 Model design

When playing chess, the simplest idea is to estimate where the opponent will be able to play the next step when own size chose to move the chess disc to this position, and the impact on the next move of the opponent and even on the whole chess game.

The same is true of applying this idea to computers. Data structure *tree* can be used to simulate the state of a chess game, like Fig2.4. The current situation on the chessboard is placed in the root node. If take a move from the current situation, the root node will generate a new branch. When continue to walk one step, it will continue to generate new branches in the new node of the new branch.

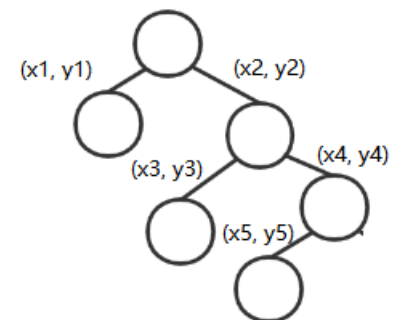


Fig2.4

It would be a good idea to use a computer to estimate the result of each step, and then choose a better step.

But for Reversi, there are too much steps in the game that computer can't estimate every step. However, although it is impossible to estimate the moves of each outcome, computer can estimate the possible moves of the next few steps and the chess game obtained for the current chessboard. A feasible solution is to evaluate whether the chessboard is beneficial to own size and the extent to which it is beneficial.

2.3.1 Depth-limited Search Algorithm

When the game state is encoded into the data structure *tree*, the depth limited search algorithm can be used to find the chessboard state of the *depth* (such as 5) layer nodes and evaluate them to find the most favorable chessboard for own size.

The depth limited search algorithm seems to have selected a favorable move, but there is a problem. The opponent does not necessarily follow this path to make the chessboard to the specified situation. On the contrary, the opponent wants to win and will try best to avoid this path.

2.3.2 Minimax Search Algorithm

Minimax search algorithm considers situations where both parties are as smart as possible. When own size play chess, choose a step that is relatively favorable to own size, and when opponent play chess, choose a step that is less favorable to own size.

Like Fig2.5, $V_{\text{node2.1}} = 3$, $V_{\text{node3.1}} = 4$, $V_{\text{node3.2}} = 2$, $V_{\text{node2.3}} = 2$, $V_{\text{node2.4}} = 1$.

$$V_{\text{node2.2}} = \max(V_{\text{node3.1}}, V_{\text{node3.2}}) = 4$$

$$V_{\text{node1.1}} = \min(V_{\text{node2.1}}, V_{\text{node2.2}}) = 3, V_{\text{node1.2}} = \min(V_{\text{node2.3}}, V_{\text{node2.4}}) = 1$$

$$V_{\text{node0}} = \max(V_{\text{node1.1}}, V_{\text{node1.2}}) = 3$$

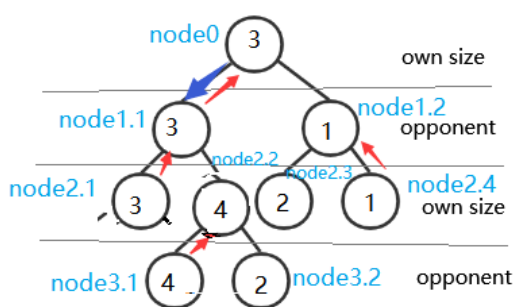


Fig2.5

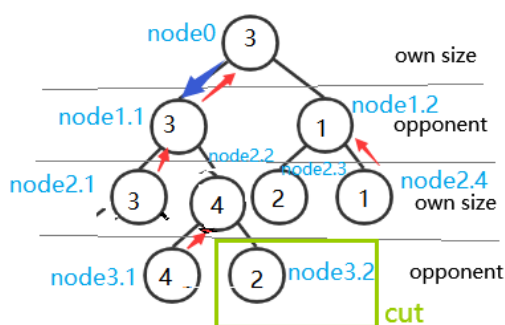


Fig2.6

2.3.3 Alpha-Beta Pruning Algorithm

When use the minimax search algorithm, some branches can be removed. It avoids wasting time on unnecessary branch searches

After get the node2.1 value is 3, node3.1 value is 4 which is large than node2.1, it is obvious:

$$V_{\text{node2}} = \max(V_{\text{node3}}, V_{\text{node4}}) \geq 4$$

$$V_{\text{node5}} = \min(V_{\text{node1}}, V_{\text{node2}}) = V_{\text{node1}}$$

It means that there is no need to search the branch where node4 is located.

2.3.4 Iterative Deepening Search Algorithm

There are many kinds of chessboard states, how to choose a better search depth is a problem worth considering for a given search time. If the depth is relatively large, it may not be able to search all steps in a limited search time, which will lead to inaccurate evaluation results. In addition to selecting a smaller depth to ensure that the depth can be searched, there is a feasible scheme to use the iterative deepening algorithm, which will use the given search time as much as possible.

Set a relatively small search *depth* at first, and judge whether it is about to time out during the search. If so, terminate the search in time, otherwise, the *depth* will continue to increase and continue the search.

2.3.5 Design of evaluation function

Chessboard evaluation is the most important part of the Reversi search algorithm. The quality of the evaluation often affects the trend of the chess game.

In the Reversi chessboard, the corner(notation: a1, h1, a8, h8) is a great position. If the corner is obtained, it cannot be flipped again. Therefore, when assigning the weight of the chessboard, the weight of the corner is often assigned a larger value. In contrast, the weight of the positions (X-squares and C-squares) where the opponent has an easy chance to obtain a corner will be small. In addition to the corners, the positions on the edges are also good choices. These positions can only be flipped by the chess pieces on the edges.

The position that cannot be flipped again is called the stable discs, and the number of stable discs can be used as one of the criteria for chessboard evaluation.

Another important evaluation indicator is mobility, which refers to the number of positions that can be falling disc, which is usually more important in the middle of the game.

The Reversi game is constantly changing, and the above-mentioned indicators play different roles in different stages, so different adjustments are needed at different stages.

In general, at the beginning of the game, strive to obtain the corner earlier to lay the foundation for increasing the number of stable discs later. At this time, the weight of the chessboard corner should be taken seriously. In the middle game, mobility provides more opportunities. By the end of the game, neither side player will have much mobility, and it will be a good choice to pursue the number of flipped discs.

The definition of the chess game state can be the number of chess discs on the board, or the number of chess discs on the border, or how many steps the player took, etc.

2.4 Details of algorithms:

Function call example: Minimax Search Algorithm: $\text{value} = \text{MinMax}(5)$,

Alpha-Beta Pruning Algorithm: $\text{value} = \text{AlphaBeta}(5, -\text{INF}, \text{INF})$

Below is the pseudo code of the algorithm.

- Minimax Search Algorithm

```
int MinMax(int depth) {  
    if ( CurrentMove == OwnSize ) {  
        return Max(depth);  
    } else {  
        return Min(depth);  
    }  
}
```

```
int Max(int depth) {
    int best = -INF;
    if (depth == 0) { return Evaluate(chessboard, CurrentMove); }
    GetNextMoves();
    while (MovesLeft()) {
        MakeNextMove();
        value = Min(depth - 1);
        UnmakeMove();
        if (value > best) { best = value; }
    }
    return best;
}
```

```
int Min(int depth) {
    int best = INF;
    if (depth == 0) { return Evaluate(chessboard, CurrentMove); }
    GetNextMoves();
    while (MovesLeft()) {
        MakeNextMove();
        value = Max(depth - 1);
        UnmakeMove();
        if (value < best) { best = value; }
    }
    return best;
}
```

- Alpha-Beta Pruning Algorithm

```
int AlphaBeta(int depth, int alpha, int beta) {
    if (depth == 0) {
        return Evaluate(chessboard, CurrentMove);
    }
    GetNextMoves();
    while (MovesLeft()) {
        MakeNextMove();
        value = -AlphaBeta(depth - 1, -beta, -alpha);
        UnmakeMove();
        if (value >= beta) {
            return beta;
        }
        if (value > alpha) {
            alpha = value;
        }
    }
    return alpha;
}
```

- Iterative Deepening Search Algorithm

```
depth = Initial
while ( true ) {
    value = AlphaBeta(depth, -INF, INF);
    depth ++;
    if (TimedOut()) {
        break;
    }
}
```

- Evaluation function

```

int Evaluate( chessboard, color ):
    weightBorad = [[20  2 12 16 16 12  2 20]
                    [ 2  0  5  4  4  5  0  2]
                    [12  5  3  2  2  3  5 12]
                    [16  4  2  2  2  2  4 16]
                    [16  4  2  2  2  2  4 16]
                    [12  5  3  2  2  3  5 12]
                    [ 2  0  5  4  4  5  0  2]
                    [20  2 12 16 16 12  2 20]]

    if ( CurrentStage(chessboard) == BeginStage ){
        actionWeight = 100
        weightWeight = 50
        stableWeight = 30
    }else if ( CurrentStage(chessboard) == MiddenStage ){
        actionWeight = 150
        weightWeight = 60
        stableWeight = 50
    }else{
        actionWeight = 80
        weightWeight = 45
        stableWeight = 80
    }

    value = weightWeight * sum(weightBorad * chessboard)
           + actionWeight * GetNextMoves(chessboard, color)
           - actionWeight * GetNextMoves(chessboard, -color)
           + stableWeight * GetStableCount(chessboard, color)

    return value

```

This is one way to implement function *CurrentStage(chessboard)*

```
int CurrentStage(chessboard){  
    count = The number of squares on the board with no disc  
    if ( count > 50 ){  
        return BeginStage  
    }else if( count > 10 ){  
        return MiddenStage  
    }else{  
        return EndStage  
    }  
}
```

3 Empirical Verification

This section is about the verification of experiments

3.1 Dataset

In the usability test, I used the test cases that the course provided, and found problems in the usual battles, it is enough. For the points race and the round robin, I relied on the Reversi platform to allocate an opponent for me. At the end of each game, I'll go back to the game process find out what's wrong with my program, and then improve it. In particular, I also wrote down some special test samples. When modified the program, I will test them to check if their results are reliable. I used the software WZebra [2] to help me evaluate which step I should take next.

3.2 Performance measure

I use the course Reversi platform to test the performance of the program. Compared to randomly selecting a position to fall disc, the program written with a designed algorithm can defeat more opponents. After a series of optimizations (such as avoiding double counting lead wasting time), my score ranking has been improved.

3.3 Hyperparameters

The parameters involved in this project are the depth of search, the weight of the chessboard weight, the weight of the mobility, and the weight of the stable discs, estimation of chessboard position.

For the depth of search, 3 was chosen in the early days. Later, search depth dynamic adjustments in the program according to the mobility. When the mobility is relatively small, search depth was 4, when the mobility is relatively large, search depth was 2, otherwise search depth was 3. Finally, the Iterative Deepening Search Algorithm was used, so that the result is at least not worse than the original.

For weight parameters, when the game changes, they are unchanged in the early

days. Later adjusted according to the stage of the game.

For estimation of chessboard position, they are also unchanged in the early days, but this is a little problematic. When the corner was occupied, the estimation of X-squares and C-squares should be changed. Later program dynamic adjustment X-squares, C-squares, and edge position's estimation.

3.4 Experimental results

All test cases were passed in the usability test, including special tests and normal battles.

The ranking of the points race from more than 150 to 70, and finally fell back to about 100 and fluctuated up and down. The final round robin ranking is 133.

3.5 Conclusion

My algorithm used the available time as much as possible to achieve a certain combat effect. However, the parameter adjustment has not been done well enough, and the evaluation factor of the evaluation function is not enough.

From this project, I have a better understanding of search algorithms and their application in Reversi. I learned the strategy of replacing time (the time required for the program to find the result) by using space (the use of program memory).

In the improvement of the algorithm, the focus is on the evaluation function. In addition to adjusting the parameters, it is worth considering adding more evaluation factors, such as unbalanced edge, internal discs, external discs, etc.

References

- [1] R. Brian, "Chapter 1 Rules and notation," Othello: A Minute to Learn... A Lifetime to Master", 2005, pp. 1-1.
- [2] "Download WZebra 4.2.4." [Online]. Available:
<http://radagast.se/othello/download.html>.