Othello Legal Game Tree Size Calculations

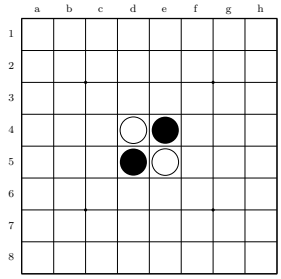
Aaron Jencks

10/21/2023

Contents

1 Background

Reversi is an age old game that was invented in 1883 and later became known as Othello in 1971. Othello differs a little from Reversi in the way that the game starts; in Reversi, the players take turns placing the first four pieces, but in Othello, the first four pieces are already placed. The figure below displays the initial board for Othello.[**wiki**]



1.1 Gameplay

The player with the dark pieces always goes first. The game proceeds as players take turns placing new pieces onto the board, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are flipped to the current player's color. Whichever player has the most disks of their color when either the board is filled, or there are no more legal moves, wins. There is only one additional rule, each play must capture/flip at least one piece.[**wiki**]

2 Introduction

Many advances have been made in the world of chess computers since the late 1900s, but less have been made in the field of Othello computers. The game has been solved, the current popular implementation is known as logistello which has beaten the human champion 6-0. This implementation uses modern tree search methods and was developed by Michael Buro.[**logistello**]

I've decided to improve on the statistics of the game. It is estimated that the number of positions in the game is 10^{28} but I hypothesize that the number of legal positions is much less. So, I've made a graph walker to count the number of legal positions that exist in the game.[**wiki**]

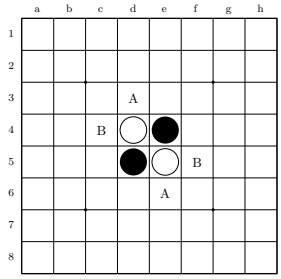
3 Methodology

3.1 Game Symmetry

With the search space of the problem being so large, I needed to find ways to reduce it. Luckily the entire game is symmetrical.

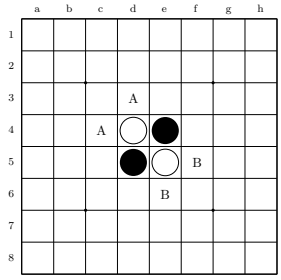
3.1.1 Rotation

There are four possible starting positions. Any game started from any of the 2 positions from one corner can be moved to a corresponding move started from a different starting position by rotating all of the disks on the board 180 degrees. See the figure below. In the figure below each letter can be rotated to by the other square of the same letter.



3.1.2 Reflection

The other way that there is symmetry in the board is by using reflection across the central diagonal. Each game started from a corner of the initial four pieces can be converted to a game that started on the other square that shares the same corner, by reflecting the game board across the diagonal. You can imagine this is like if we rotated the board by 90 degrees and then flipped the board along the horizontal axis. See the figure below. In the figure below each letter can be rotated to by the other square of the same letter.

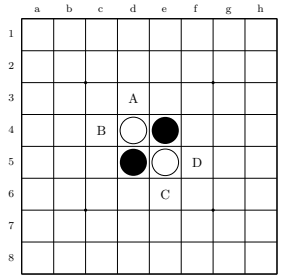


3.1.3 Reversal

The last way that there is symmetry in the game is that we can also invert the colors of the board. In this case all of the light pieces would become dark and all of the dark pieces would become light. This would be useful if either player could start the game, but since the rules state that the dark player always begins, there is no use for this symmetry right now. This transformation is equivalent to rotating the board by 90 degrees.

3.1.4 Full Circle

Using first 2 methods, I can show that, I can convert any game from any corner of the initial position to any other corner of the initial position, see below:



For simplicity's sake, I'm only going to use a single corner as an origin

- $A \rightarrow B$ Reflection
- $A \rightarrow C$ Rotation
- $A \rightarrow D$ Rotation, Reflection

Because any position from A can be mapped to B and any position from B can be mapped to A, then that means that A and B are fully mappable, this means that all games in A have corresponding games in B and vice versa. The same is true for all letters to each other letter. Because of this, I can do a full traversal of one opening move and find all possible games in the entire game tree by mapping the moves to the other 3 corners.

3.1.5 Additional Uses

The value of symmetry to reduce state space in this simulation is infallible. While its use was limited to only the first move in this simulation, there may exist further uses to reduce search tree exploration during game play. One must consider the cost of performing these symmetry checks, versus just traversing the tree.

3.2 Implementation

The overall implementation of this simulation is relatively simple. There is a main routine that contains a set of global variables that are updated as one or more walkers traverse the game space given some initial starting board. Once all of the walkers exit, then the main routine reports those variables to the terminal and those numbers are your counts. Complexity arises in this project not due to it's simulation, but because of the sheer size of the game space.

3.2.1 Board Representation

Because there are going to be several board running at any given time, care must be taken to avoid consuming unnecessary amounts of memory, because of this a memory efficient representation of the board was designed. It allows using a one dimensional array of bytes with a size of 16 bytes. Each byte represents 4 cells, 2 bits each, and each row contains 2 bytes. The get and put methods of the boards then parse this array of bytes to manipulate and read the data from within using bit manipulation operators.

Hashing There is additional care that must be taken to hash the boards, since not only do the piece locations matter when comparing boards, but also which player's turn it is. To get the board state to fit within a single integer, additional optimizations were used. This optimization was to reduce the bits used for the center 4 squares. Since the center 4 squares can never be empty, they only need 1 bit each, this eliminates 4 bits from the overall requirement. If we assume that we're using an 8x8 board, then our bit requirement comes down to $2 * 60$ bits for the non-center cells plus 4 bits for the center cells plus 1 bit for a player indicator. This comes to 125 bits. To fit in a standard integer size, this is rounded up to 128 bits.

Spiral Hashing To increase locality amongst the cache, to reduce memory bandwidth usage. A spiral hash was used. This hash starts with the center 4 squares, then spirals in a counter-clockwise fashion outward towards the outer edge of the board. This helps keep similar boards close to each other in hash value, so that they can be placed in closer bins, hopefully in the same page of memory.

3.2.2 Checkpoint Files

Because the simulation can take multiple days to finish for anything bigger than a 4x4 board, a checkpoint system was implemented. This stores binary data that can be restored from so that the simulation can be stopped and started without losing progress. The layout of the file is as follows:

<table>
<tr><td colspan="16">Checkpoint File V1</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>15</td></tr>
<tr><td colspan="8">$L_{version}$</td><td colspan="8">$L_{version}$ bytes Version String</td></tr>
<tr><td colspan="8">Counted</td><td colspan="8">Explored</td></tr>
<tr><td colspan="8">Repeated</td><td colspan="8">Elapsed Time</td></tr>
<tr><td colspan="8">L_{cache}</td><td colspan="8">$L_{cache} * 16$ bytes Cache Contents</td></tr>
<tr><td colspan="16">Rest of file: Walker Stack Contents</td></tr>
</table>

Additional Checkpoint Notes The contents of the last two sections of the diagram above contains board hashes, these are 16 byte representations of boards. You can see the section on Hashing Boards for more information. The first section (the cache contents) has a length parameter given before it, this specifies how many uint128 values belong to the cache before the rest of the file is dedicated to the walker stack contents (the boards waiting to be explored for DFS).

4 Conclusions

4.1 4x4 Board

The total possible positions in Othello for a 4x4 board are $8,503,056$.[**wiki**] The total number of legal moves possible in Othello on a 4x4 board is $34,400$. And the total number of final board states possible is $4,604$.

4.2 6x6 Board