Othello Heuristics for Evaluation Functions

Vlad-Mihai Bogdan

April 8, 2024

Introduction

Thing to consider when designing an evaluation function

Othello is a game with a $8x8$ board, so the number of possible games can pe weakly estimated to be 3^{64}. This means that the search space is huge, and the evaluation function is crucial for the performance of the engine.

Creating a good heuristic is a difficult task, as it requires a deep understanding of the game. And since I don't have, I will just try to focus on the most important aspects of the game.

The factors that I will consider are:

- Mobility
- Stability
- Coin Parity
- Corners and the weight of each square

Mobility

Mobility is the number of legal moves that a player has. As a general rule, the less moves a player has, the worse the position is, since it means that the player may be forced to make a bad move.

Mobility comes in two forms: **Actual mobility** and **Potential mobility**. Actual mobility is simply calculated by counting the number of legal moves that a player has. Potential mobility is the number of squares that are adjacent to at least one of opponent's discs.

Stability

Stability is the number of discs that can't be flipped. Such discs are called **stable discs**. For example, the corners are stable discs, since they can't be flipped. Since the goal of the game is to have as many discs as possible, knowing the number of discs that will defenetly be yours is important. We will call a disc unstabile if it can be flipped in the next move.

Coin Parity

Coin Parity is the difference between the number of discs of the player and the number of discs of the opponent. This is defenetly as important as the other factors, since a single move can flip up to 18 discs.

Corners and the weight of each square

The **corners** are the most important squares in the game, since they are stable. This gives us the ideea that the neighbouring squares of the corners are not good to capture, as they would lead to the loss of the corner. Using this ideea, we will assign a weight to each square, based on its importance.

Static weight evaluation function

As we said, different squares have different importance. For a simple and fast evaluation function, we will assign a weight to each square, based on its importance. The result of the function will be something like this:

$$eval(b) = \begin{cases} -\infty & \text{if MIN won} \\ \infty & \text{if MAX won} \\ \sum_{(i,j) \in b_{MAX}} weight_{i,j} - \sum_{(i,j) \in b_{MIN}} weight_{i,j} & \text{otherwise} \end{cases}$$

where b_{MAX} is the set of cells occupied by MAX in b. Same for b_{MIN}.

The weights of the squares I will be using are:

$$\begin{bmatrix} 99 & -8 & 8 & 6 & 6 & 8 & -8 & 99 \\ -8 & -24 & -4 & -3 & -3 & -4 & -25 & -8 \\ 8 & -4 & 7 & 4 & 4 & 7 & -4 & 8 \\ 6 & -3 & 4 & 0 & 0 & 4 & -3 & 6 \\ 6 & -3 & 4 & 0 & 0 & 4 & -3 & 6 \\ 8 & -4 & 7 & 4 & 4 & 7 & -4 & 8 \\ -8 & -24 & -4 & -3 & -3 & -4 & -25 & -8 \\ 99 & -8 & 8 & 6 & 6 & 8 & -8 & 99 \end{bmatrix}$$

Mobility-based evaluation function

The static weight evaluation function is good, but it doesn't take into account the number of discs on the board and the other factors discussed above.

So, first of all, we will divide the function into two parts:

$$eval(b) = \begin{cases} early\_eval(b) & \text{if less than } TH\_EARLY \text{ moves have been played in } b \\ late\_eval(b) & \text{if more than } TH\_LATE \text{ moves have been played in } b \\ mid\_eval(b) & \text{otherwise} \end{cases}$$

where TH_EARLY and TH_LATE are thresholds for deciding when to switch from early to late evaluation.

Both of these functions will be a linear combination of the factors discussed above. We will use $O\_atribute$ to describe the value of the attribute for the opponent, and $P\_atribute$ to describe the value of the attribute for the player.

Early evaluation

The main difference between the early and the mid/late game is that in the beginning of the game, it could be better to have less discs on the board, since this will give the opponent less moves to choose from. This phenomenon is called **evaporation**.

So, for taking this into account, we will use the following formula:

$$\text{coin\_parity} = 100 \cdot \frac{O\_discs - P\_discs}{O\_discs + P\_discs}$$

We now have to consider mobility:

$$\text{mobility} = 100 \cdot \frac{P\_mobility - O\_mobility}{P\_mobility + O\_mobility}$$

$$\text{potential\_mobility} = 100 \cdot \frac{P\_potential - O\_potential}{P\_potential + O\_potential}$$

And the stability:

$$\text{corners} = \begin{cases} 100 \cdot \frac{P\_corners - O\_corners}{P\_corners + O\_corners} & \text{if } P\_corners + O\_corners \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

As we said, the early evaluation function will be a linear combination of these factors.

Mid evaluation

We will be using the same factors as in the early evaluation function, but we will change the weights of the factors. We will also ignore the evaporation factor, since it is not important in the mid game. The focus will be on the mobility of the player and the stability of the discs.

Late evaluation

In the late game, the most important factor is the number of discs that each player has, so we will focus on the coin parity. The function pretty much stays the same, but the weights of the factors will be different.

Further possible improvements

When talking about the mobility-based evaluation function, we can also divide the mid and late game into two separate functions. The function evaluation a late game position can try to reach more leaf nodes and take into who will make the last of the game (since this move adds some discs that cannot be flipped).

Another possible improvement is to use a the ProbCut/Multi-ProbCut algorithm, which is a variation of the alpha-beta pruning algorithm. The algorithm uses a threshold to decide when to stop searching a branch of the tree, and it is based on the ideea that the best move is usually close to the root of the tree. The algorithm was shown to be very effective in practice, and it is used in many Othello engines.