Othello

Amina A. Aljubran

Indiana State University

aaljubran@sycamores.indstate.edu

December 12, 2014

Abstract

Many different algorithms of playing games have been suggested which is considered artificial intelligence. Othello is one of two-player games which is PSPACE-COMPLETE 1 problem. There are AI algorithms that computers depend on to decide where the computer to play. The algorithms that are discussed in this paper are MiniMax and Alpha Beta Pruning algorithms.

1 Introduction

Today, we are living in the age of technology. Technology is almost used for everything encounter people in their life. Artificial intelligence is the field of computer science which concerned with making computers corresponded as humans. Playing game is one of the popular artificial intelligence branches because of its applications to plenty of real world problems.

Mostly, games depend on a few simple rules that are not hard to be followed by a computer. One of these games is Othello which was invented by Goro Hasegawa in 1971. He was inspired by the Chinese game "Go". Othello has different names such as Reversi and Annex. In 1973, Tsukuda Original Company produced Othello in Japan. Nowadays, the right of Othello is owned by Anjar Company[Men08]. However, there are plenty of Othello programs on the Internet which are free. These games are built by using a wide variety of searching and timing techniques. In this game, players flip discs each time they play. Human players cannot imagine the changing in the discs board that are occurred by a computer moves. On the other hand, computers are predictable for both humans and computers moves by using AI search algorithms.

In this paper, there are descriptions for Minimax and Alpha-Beta Pruning algorithm. There are also pseudo code and the running time for both of them. The evaluation technique which is used in the project programs is payoff technique 2.

1.1 Organization

In Section 2 some of the terms, which are used, are defined. In Section 3 the algorithms that are studied are explained by using pseudo codes and examples while In Section 4 there are the running time for each algorithm. In Section 5 there is the result that got from the project. In Section 6 It contains future work suggestions.

2 Definitions

Definition 1 (PSPACE-COMPLETE). *The problem is PSPACE-COMPLETE if it can be solved using an amount of memory that is polynomial in the input length (polynomial space).*

Definition 2 (Payoff Technique). *evaluation technique depends on the discs number which is represented by subtracting players' discs.*

3 The Algorithm

Othello depends on zero-sum property that is a first player's gain (or loss) of discs is exactly equal to the number of discs losses (or gains) of the other player. In another words, If the gains number of discs of the winner subtracted from the number of discs which the loser lost, they will be zero. Strictly competitive game is another name of a zero-sum game. This kind of game is most often solved by using minimax and Alpha-Beta Rruning algorithm.

3.1 MiniMax Algorithm

3.1.1 Background

This algorithm is considered the heart for many algorithms which have been used for a Sum-Zero game. In 1928, the theorem of this algorithm was published by John von Neumann[TL59]. Neumann is cited as saying "There could be no theory of games .. without that theorem .. I thought there was nothing worth publishing until the Minimax Theorem was proved[Cas96]." Minimax algorithm is for fully observable and deterministic games. These games include classic board games such as chess, checkers, tic tac toe, and go.

3.1.2 Explanation

MiniMax is a recursive algorithm that searches for any possible legal moves in order to find the move which produces the best result. This algorithm depends on Depth-First search (DFS) algorithm. MiniMax assumes that there are two players who are called Max and Min. When it is Max's player turn, it chooses the move that maximizes his gain while it chooses the move that minimizes the Max player gain on Min's player turn. It is minimizing the possibility of losing the game for a worst case which is maximum loss scenario. On another words, Minimax is denote to minimize the maximum value of the opponent's possible moves.

3.1.3 Pseudocode

Code

```
function MiniMax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        bestValue := -infinite
        for each child of node
            bestValue := max(bestValue, minimax(child, depth - 1, FALSE))
        return bestValue
    else
        bestValue := +infinite
```

```
        for each child of node
            bestValue := min(bestValue, minimax(child, depth - 1, TRUE))
        return bestValue

(* Initial call *)
minimax(origin, depth, TRUE)
```

Suppose that there are two players Max and Min plays a Sum-Zero game. The algorithm generates a depth-first search tree. It starts from the current game position up to the last depth or the end of game. Then, the final node returns its value which is compared to the previous returned value (bestValue). If maximizingplayer is TRUE, It will return the maximum value of the children nodes. Otherwise, It will return the minimum value of them.

3.1.4 Example

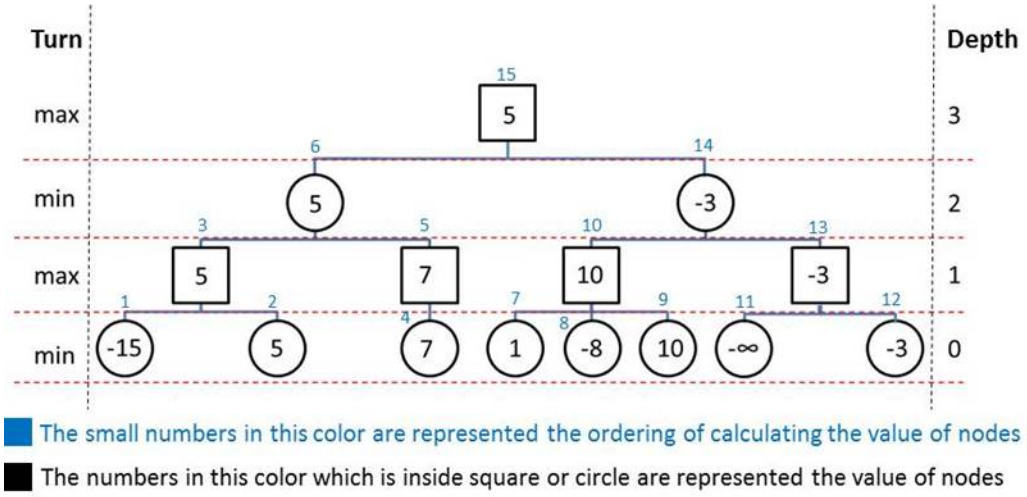

Figure 1: MiniMax example

Each node of the tree is filled bottom-up with the evaluated values. The first evaluation happens to the node of the last depth or the terminal one. The value of each node is associated with each discs positions in the game. In the above example (Figure-1), The algorithm evaluated node "1" then returned the value to node "3". Afterward, It evaluated node "2" and returned it. When the value of node "2" is returned, The algorithm compared the value of node "3" with the value that returned and chose the largest value of them which is 5. After that, The value of node "3" stored in node "6" then filled node "4" which is the node of the last depth. While there is no more than one node, the value of node "4" is stored in node "5". In this case, the Algorithm chose the value 5 instead of 7 where It is Min's turn. The evaluating of the children nodes are continued alternately until it reached the root node. The root node is Max's turn which always choose the move with the largest value.

3.2 Alpha Beta Pruning Algorithm

3.2.1 Background

Minimax algorithm searches entire nodes, even if it is obvious that some nodes of the tree can be pruned ignored. There is an optimization which improves MiniMax algorithm is called Alpha-Beta-Pruning algorithm. Instead of searching entire tree, Alpha-Beta-Pruning is a technique for pruning nodes that are not needed to evaluate the possible moves. This will save a lot of searching time which allow increasing search depth. In 1961, McCarthy proposed Alpha-Beta during the Dartmouth Conference[Kot04].

3.2.2 Explanation

In this algorithm, Alpha represents the best position value which Max player can reach by using a pessimistic evaluation while Beta represents the best position Max player can reach by using an optimistic evaluation. The value of alpha in the beginning of search is -infinity while beta is +infinity. At each depth, the algorithm uses optimistic and pessimistic to evaluate the nodes then compares between alpha and beta. If a Max player's move is optimistically less than alpha, it is ignored because of a better alternative exists. Also, if a Max player's move pessimistically is better than beta, it is pruned because of Min player had a better alternative exits. This algorithm saves searching time without affecting on the final result.

3.2.3 Pseudocode

Code

```
function AlphaBeta(node, depth, alpha, beta, maximizingPlayer)
    if depth = 0 or node is a terminal node
         return the heuristic value of node
    if maximizingPlayer
         for each child of node
             alpha := max(alpha, alphabeta(child, depth - 1, alpha, beta, FALSE))
             if alpha >= beta
                 break
         return alpha
    else
         for each child of node
             beta := min(beta, alphabeta(child, depth - 1, alpha, beta, TRUE))
             if beta <= alpha
                 break
         return beta

(* Initial call *)
AlphaBeta(origin, depth, -infinity, +infinity, TRUE)
```

3.2.4 Example

It might be the best way to explain this algorithm by going through a example and follow the algorithm to evaluate the nodes.

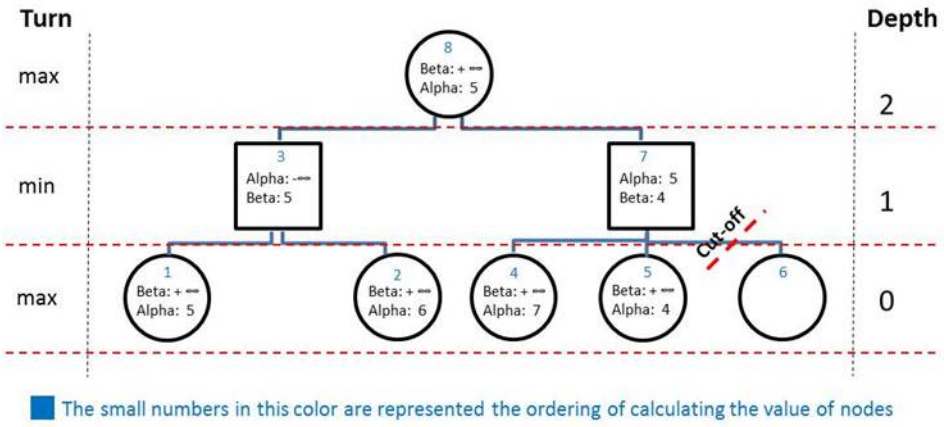

Figure 2: Alpah Beta Pruning example

The explanation as the algorithm is running

```
node "8"
        alpha = -infinity , beta = +infinity
        node "3"
                alpha = -infinity, beta = +infinity
                node "1"
                       alpha = 5
                beta = min(+infinity, 5) = 5
                if(5 <= -infinity): No
                node "2"
                       alpha = 6
                beta = min(5, 6) = 5
                if(5 <= -infinity): No
        alpah = 5

        node "7"
                alpha = 5, beta = +infinity
                node "4"
                        alpha = 7
                beta = min(+infinity, 7) = 7
                if(7 <= 5): No

                node "5"
                        alpha = 4
                beta = min(7, 4) = 4
                if(4 <= 5): yes
                 (break the loop. Do not evaluate the other children of node "7")
        alpah = max(5, 4) = 5
```

4 Runtime

In MiniMax algorithm, the evaluation happens to each node in the tree because of that the runtime could be represented by using branching factor and search depth. If the search depth of tree is d and the constant or average of branching factor is b, the runtime will be

$$O(b \cdot b \cdot ... \cdot b) = O(b^d)$$

which is the maximum number of nodes that are evaluated. Notice that in Othello there is no specific number for branching factor since it depends on the possible legal moves.

Alpha Beta Pruning avoids processing nodes that have no effect on the final result. In the best case, if the best legal moves (evaluation) were always searched first, the number of nodes which would be evaluated will be about the half of the number of legal moves[HIP14]. In this case, the algorithm needs to find the best Mix's player moves while needs to only the best Min's player move to refute all other nodes.

$$O(b^{\frac{d}{2}}) = O(\sqrt{b^d})$$

The worst case will be when the nodes are in pessimistic order which means the best value of Min's player is checked in the last node. The runtime in the worst case is equal to the run time to MiniMax algorithm.

$$O(b \cdot b \cdot ... \cdot b) = O(b^d)$$

The average case will be when the nodes are ordered at random which is approximately

$$O(b^{\frac{3d}{4}})$$

5 Our Contribution

The database of the games that is played between strong players could be used as input for Pattern-based technique.

6 Future Work

The future research will be about different algorithms like Negascout, MTD-f, and NegaC. These algorithms could be applied on Othello with different evaluation techniques such as Disk-square tables, Mobility-based, and Pattern-based.

References

[Cas96] John L Casti. *Five golden rules: great theories of 20th-century mathematics, and why they matter. p.19.* Wiley-Interscience, NY, USA, 1996.

[HIP14] Jaap Herik, Hiroyuki Iida, and Aske Plaat. *Computers and Games: 8th International Conference, p.119.* springer International Publishing Switzerland, Yokohama, Japan, 2014.

[Kot04] Alan Kotok. Artificial intelligence project-mit, a chess playing program, 2004. www.kotok.org/AI_Memo_41.html.

[Men08] Beppi Menozzi. Brief history of othello, 2008. www.beppi.it/public/OthelloMuseum/pages/history.php.

[TL59] Albert William Tucker and Robert Duncan Luce. *Contributions to the Theory of Games, Volume IV.* Princeton University Press, Princeton, NJ, USA, 1959.