CS6380 - Artificial Intelligence (Assignment 2:Othello)

Akshay G Rao (CS21S002)

20th November 2021

1. Background

Othello is the modern version of the game ‘Reversi’ which is a board game played on an 8x8 unchecked board **[1]**. Players take turns placing disks on the board with their assigned color facing up. During a play, 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 turned over to the current player's color. The objective of the game is to have the majority of disks turned to display one's color when the last playable empty square is filled.

A game is said to be solved if we can predict with full certainty the outcome of the game even when both players play perfectly. So in such games, just knowing the current state of the game is enough to conclude the winner of the game even when the opponent played perfectly. The 4x4 and 6x6 variants of Othello are fully solved since all the possible game states and their outcomes can be computed. But in the case of the default 8x8 Othello, the possible games are too big to be fully explored and solve fully. However, the “open book”(the best few sets of well-known moves) can be maintained to quickly play the first few moves of the game optimally and save computation.

2. Strategies for game playing

a. Minimax algorithm with alpha-beta pruning[2]

The game of Othello is complete information (both player know each other states completely and the set of moves available completely), zero-sum (outcome of the game is draw, win or lose), and a two-player game. Hence minimax algorithm would be applicable for this setting. A game tree is constructed containing alternative MAX and MIN layer. MAX layer represents the game from the current player’s perspective and MIN layer represents the game from the opponent player’s perspective. Hence the current player(MAX) chooses the best node which maximizes the overall outcome of the game assuming that the opponent plays at his best too(i.e opponent tries to drive the game to minimum value). Hence this algorithm considers game playing by taking into account the decision of both the player in an alternative fashion.

Alpha-beta pruning is a strict and guaranteed(i.e it prunes parts of game tree which doesn’t influence the value of the game while using minimax algorithm) optimization strategy over the minimax algorithm. It avoids examining parts of the game tree when it is known that a better alternative is already available. This is done by maintaining the alpha and beta window.

b. K-Ply search with evaluation function over the horizon

The above alpha-beta pruning algorithm assumes that the entire game tree can be examined and hence the outcome of the game is known at the leaf which can be backed up using the minimax algorithm. But in practical scenarios, it is impossible to evaluate the whole game tree due to its enormous size. Hence we need to construct a game tree only up to a certain depth and use a heuristic to evaluate the game state at the leaf. This is popularly known as K-ply search wherein we evaluate the game with depth k. For evaluating the game state**[3][4]** at the horizon, the following factors were considered:

1. Mobility
2. Corners captured
3. Corner closeness
4. Number of frontiers
5. Coin parity
6. Positional advantage table

c. Alpha-beta pruning with iterative deepening[5]

In this assignment, the time constraint imposed was 2 seconds per move for each player. The game would have a fewer number of possibilities at the beginning, a very large number of possibilities in the mid-game, and again fewer number of possibilities in the end-game. Hence the mid-game would act as a bottleneck in determining the value of k(maximum lookahead depth) in the k-ply search. Hence fixing a particular value of k would not utilize the whole time provided and also fixing a k might be dependent on the system which runs the algorithm. Hence we can increase the depth of the k-ply search iteratively so that when a timeout occurs we have a good move to return instead of diving into huge depth with no results when a timeout occurs.

d. Alpha-beta pruning with iterative deepening and transposition table

Using iterative deepening increases the time complexity a lot since we are re-evaluating many nodes repeatedly over different depth iterations. To avoid this re-evaluation we can use a transposition table**[6]**. This hash table(one way to implement transposition table) uses current board representation(64bit) as the key and Object (containing depth, flag, value as attributes) as the value. The flag attribute in the object holds whether the value indicates alpha value, beta value, or the value of the evaluation function. It is important to note that when a particular board representation is re-visited, one has to consider the look-ahead depth being considered to return the hashed value. If the same board is revisited with the same lookahead depth as in the object, then we can simply return the value from the hash. But, if a deeper lookahead is

about to be done, then the value in the hash cannot be used even though the board position is the same.

3. Conclusion

I have used alpha-beta pruning with iterative deepening and transposition table for game playing here. Also the board evaluation heuristic was quite sophisticated (as described in 2. b). The algorithm is very well suited for a time constraint setup since it has the best move at all times and improves the move if more time is remaining. This ensures that the algorithm is independent of the system execution speed and the phase(opening, mid-game, and end-game) the game is currently in.

4. Other possible strategies

One can try the SSS*[7] algorithm. But generally, the SSS* is very resource-intensive and under-performs compared to alpha-beta pruning in a very time-constrained setup. So another possibility is the usage of the negamax algorithm **[8]**. Three variations of SSS* which performs better than alpha-beta pruning (in some cases) are available in reference **[9]**. However, I didn't use any of these because we have a very strict time constraint here.

References

[1] "Reversi". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Reversi.
[2] Alpha-beta minimax algorithm (https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/)
[3] Evaluation strategies http://www.samsoft.org.uk/reversi/strategy.htm
[4] An Analysis of Heuristics in Othello by Vaishnavi Sannidhanam and Muthukaruppan Annamalai (https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf)
[5] Alpha-beta pruning with iterative deepening (http://web.archive.org/web/20071214141335/http://www.seanet.com/~brucemo/topics/iterative.htm)
[6] Transposition table (http://web.archive.org/web/20080315233307/http://www.seanet.com/~brucemo/topics/hashing.htm)
[7] SSS* algorithm (https://en.wikipedia.org/wiki/SSS*)
[8] Negamax algorithm (https://en.wikipedia.org/wiki/Negamax)
[9] A Minimax Algorithm Better than Alpha-Beta? No and Yes (https://arxiv.org/ftp/arxiv/papers/1702/1702.03401.pdf)