CS6380: Artificial Intelligence

Assignment-2 | Report

Name: Nithin Uppalapati **Roll:** EE18B035

Name: Sagar Reddy **Roll:** CS18B029

Date: 16 Nov. 21

Introduction and Problem Statement:

Environment: See *sections 3* and *4* in the assignment problem statement pdf file

Challenges in Solving the Problem

This problem's game tree will explode to astronomical numbers (for final position of board there are around 2^{64} moves!!) and hence the challenge, otherwise in games like ticktacktoe the game tree can be analysed pretty quickly hence the game will always end in draw. Similarly in this case too if we have a "*ultimate*" computer, it is possible to analyse the game tree and will always end in draw.

So, as the game tree cannot be completely analysed, the problem now boils down to:

1. To how much depth one can explore the game tree, in a single turn of the player.
2. Pruning some of the possible strategies in the game tree
3. Evaluating Function
4. Deception Strategies

Below is the documentation of the implementation of Algorithms and Techniques for the Othello Bot.

Min-Max Backup Strategy:

The Minimax algorithm is the most well-known strategy of play of two-player, zero-sum games. The minimax theorem was proven by John von Neumann in 1928. Minimax is a strategy of always minimizing the maximum possible loss which can result from a choice that a player makes. Maximax principle counsels the player to choose the strategy that yields the best of the best possible outcomes.

Method: In Minimax the two players are called maximiser and minimizer. The *maximiser* tries to get the highest score possible while the *minimizer* tries to do the opposite and get lowest score possible. Every board state has a value associated with it. In a given state if the maximiser has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Minimax for Two-Person Games: In a two-person, zero-sum game, a person can win only if the other player loses. No cooperation is possible.

First drawback: There exists a rule in Othello that if a player cannot make a move, then the opponent has to move. In such cases the mini-max tree has special cases of **mini-mini** or **max-max** where the parent and child nodes resemble board positions after the same player has moved.

To conclude, The mini-mini and max-max drawback has to be handled along with pruning to remove redundancies in the work as the computation of each move is exponential as depth increases.

Alpha-Beta Pruning:

The Alpha-beta pruning removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

The two-parameter can be defined as:

a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximiser. The initial value of alpha is **-∞**.
b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +**∞**.

Summary of Alpha-Beta Pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Disadvantage: The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.
It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is O(b^m).
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is O($b^{m/2}$).

Heuristic Function for Evaluating a "Score" for a given board position:

As there are 64 squares in the Othello board, we constructed a 8x8 matrix. In this 8x8.

The structure of the matrix is as follows: (8 x 8 sized matrix)

a	b	d	e	e	d	b	a
b	c	f	f	f	f	c	b
d	f	g	h	h	g	f	d
e	f	h	h	h	h	f	e
e	f	h	h	h	h	f	e
d	f	g	h	h	g	f	d
b	c	f	f	f	f	c	b
a	b	d	e	e	d	b	a

Where;

a =	139.00
b =	-24.70
c =	-49.40
d =	24.70
e =	5.94
f =	-5.94

g = 19.70
h = 3.94

- As we can see the main idea was to increase the weights of the corners and edges of the board, and decrease the weights of the places which are directly adjacent to them as putting a coin there helps the opponent to occupy a corner/edge.
- Around 40 matrices were made from scratch and stored in the script file to find the best matrix that we could use for the tournament.
- The **`eval_function`** here simply computes the difference between summation of weights of red and summation of weights of black.

Genetic Algorithm for finding better Evaluation Matrix:

Crossover: The crossover method which we used is *linear combination of two evaluation matrices*. We take two parent **`parameter_matrices`** and we linearly combine them according to their fitness.

i.e., A and B are two matrices and f1 and f2 are the fitness values respectively.

We generate only one child for every two parents:

$$\text{Child matrix} = (f1*A + f2*B) / (f1 + f2)$$

Method: The parameters for GA are: ***mutation_ prob***, ***crossover_ prob*** which are useful in defining the mutation probability and crossover probability respectively. Given a pool of parents (mating pool), we randomly pick two parents according their fitness, and then we may decide to crossover them with a prob - ***crossover_ prob***. Then after the crossover process, we can randomly pick the new generation off-springs with prob - ***mutation_ prob***. But here in our case we put ***mutation_*** prob = **0** to avoid mutation.

Fitness function is based on the evaluation method after a match as given in the assignment problem statement. (See section-4 of assignment problem statement pdf file)

And at the end of each generation we:
1. We replace few worst parents with the best children.
2. Reset the fitness scores of the members and pass on to next generation.

Setting up environment in Python to do Evaluation Parameter Optimization:

1. To make the bot play games with itself, both **`MyBot.so`** and **`RandomBot.so`** were made to read their parameters from 2 files namely **`file1.txt`** and **`file2.txt`**
2. We also tweaked the framework a little bit so that it writes the scores of each game in the **`scores.txt`** file.
3. We then wrote a python script which initially has 40 parameters(40 matrices) and takes each pair of parameters and writes them to file1 and file 2 and calls the command(using **`os.subprocess()`**) to make the bots play each other.
4. The script also maintains a points table for each of these 40 **`param_matrices`** and after all the pairwise matches are done, we create 20 children from the 40 **`param_matrices`** using the points table value as the fitness function and replace the 20 worst parents of the current generation..
5. To generate each child, we choose 2 parents with probabilities related to fitness function and then combine them in the ratio of their fitness function for the parameters to converge after a large number of generations.
6. Once we have the 40 parents of the next generation, we start the next generation by resetting the points table and call a round robin tournament again as mentioned in Point-3.

Future Work:

Deception Strategies: We can model the knowledge of experts in the form of deception strategies. Doing so, it is possible to avoid the shortcomings of the K-ply search. We can also implement certain traps to the opponents with this advantage. This can be done by recognising some particular patterns in the game and so on...

Github Link:(Framework+Script) - *https://github.com/nithin-uppalapati/othello*

References:

1. https://digitalcommons.lsu.edu/cgi/viewcontent.cgi?article=1766&context=gradschool_theses
2. http://www-cs-students.stanford.edu/~lswartz/cs221/jimmypang.pdf
3. https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/O-Thell-Us/Othellus.pdf

END