Exploring AI Strategies in Othello: A Comparative Study of Learning and Search Methods

Jack Goldsmith
UNC Chapel Hill
jackg@live.unc.edu

Aknazar Janibek
UNC Chapel Hill
ajanibek@unc.edu

Matt Vu
UNC Chapel Hill
tmattvu@ad.unc.edu

Akshan Sameullah
UNC Chapel Hill
asameull@unc.edu

Jondash Karamavruc
UNC Chapel Hill
jkaramavruc@unc.edu

Abstract

This paper explores various search and learning algorithms used to optimize in playing the game Othello. Several different competitive agents are created using different algorithms and heuristics such as Monte Carlo Tree Search (MCTS), neural networks, and depth-limited minimax with alpha-beta pruning to name some. This study reveals challenges when it comes to implementing more complex heuristics and models, which raises the question about optimization and at what point do our algorithms become ineffective.

1 Motivation and Problem Definition

When it comes to playing board games, humans have struggled to win against machines that compute the most optimal play in just a few seconds. A board game such as Othello holds full accountability on the player and the decisions they make, rather than leaving it up to chance. Othello is a two-player board game that is played on an 8x8 board. Each player, black and white, starts with 32 discs of their color. The game begins from a fixed arrangement of two white and two black discs in the center of the board. Black moves first, and the two players take turns. The objective of the game is to maximize the number of discs that show your color by the end.

In this paper, we study various search and learning algorithms machines use to perform optimal play and analyze which of these algorithms are deemed most effective by creating game-playing agents. The motivation for this paper stems from wanting to understand the complexity of decisions an agent makes, and if there is possible correlation between how effective an agent performs and the complexity of their algorithm. We analyze the applications of various heuristics and if it leads to an agent performing better or worse. Exploring this specific problem that eliminates factors such as luck allows us to gain insight on the capabilities and limitations of artificial intelligence systems in a more general context.

2 Related Work

Othello is a solved game, but only as of late 2023. The 8x8 version of the game was solved last year by a computer scientist named Hiroki Takizawa. In a paper titled "Othello is Solved," Takizawa claimed that "it is computationally proven that perfect play by both players lead to a draw" (3). Othello on an 8x8 board contains 1028 unique positions, making a solution very challenging. We also attempt various heuristics analyzed by Sannidham and Annamalai at the University of Washington (2). Heuristic such as corners captured, which places more weight on the four corners of the board since once captured, they cannot be flanked by the opponent. Another heuristic is coin parity that calculates the difference in coins between the max and min player and a greedy strategy is implemented for this heuristic.

3 Method Description

3.1 Graphical User Interface (GUI)

To visualize Othello by having agents play against each other, we created a simple graphical user interface using Pygame. This interface includes black and white discs that represent each player and a green game board. In addition, the graphical user interface shows all legal moves for the current game state that the player can make on their turn, helping players who don't know how to play Othello while showing the possible moves that the agent can make.

3.2 Infrastructure

Our entire Othello project runs on a local Python application in Visual Studio Code. We created three search agents utilizing depth limited minimax (one with alpha beta pruning), two search agents utilizing Monte Carlo Tree Search, and two Convolutional Neural Networks. Each one of these agents implemented an agent interface that contain a strategy method, which takes in the current game board, the player's turn, and a time limit. Each agent has to determine what the best move is in a given time limit. This strategy method also calls upon other helper methods that the agent needs to run its respective algorithms. In addition, The purpose of this interface was to make the agents compatible with the GUI so they could play against each other. We also had three different files that take in these agents. The first file contains a program that allows humans to play against an agent. The second file allows two agents to play one another, and the third allows for the round robin tournament.

3.3 Metrics

The main metrics that we used to evaluate our agents was the average possession count and number of wins in a round robin tournament. In this tournament, each agent plays against all other agents two times. We then calculate the wins and losses that each agent accumulates, as well as the possession count. The possession count can be defined as the difference in disc count from the agent compared to its opponent. This possession count allows us to evaluate how the agent won against its opponent, with a higher possession count typically meaning that the agent dominated its opponent and a negative possession count meaning that the agent lost.

3.4 Data

For creating supervised learning agents, we used a dataset from Kaggle called Othello Games. The dataset contains gameplay data of 25,000 games from professional players. Professional players most likely are playing the optimal move in every situation, having an AI agent notice the patterns will allow it to also play the optimal move. The data consisted of 25,000 observations, where each observation consisted of a *game_id*, a binary winner variable, and lastly a *game_moves* variable. *Game_id* was simply an identifier, and winner simply stated whether black or white won. The *game_moves* was one long string of moves that were made sequentially in the game. Each move consisted of a letter and a number like "F5" denoting where on the board a piece was added. Given that we wanted our model to be able to predict what the next move given a current state should be, this data was not in a suitable format for directly training our models. We had to transform the *game_moves* variable into a dataset where each observation at least had a *current_configuration* variable, and a *next_configuration* variable, as this is what we wanted the model to predict. Optionally there was also a *possible_moves* variable which consisted of configurations that could result from the current state. We used this transformed data for training purposes.

4 Experiment and Results

4.1 Search

We went through several iterations of search agents as we learned more about the game and possible heuristics. When we discovered that the opponent could not capture corners, we added logic to prioritize corners. We also learned that there is a reasonably sized set of cases in which possessing the board's edges can lead to capturing a corner and we gave scores to each case to prioritize the best possible cases. We also discovered heuristics experimentally by watching professional Othello games such as good moves maximize a player's future moves while minimizing the opponent's future moves. We also found out that good moves tend to be squares that are close to a player's previous move so we check those first in our tree. We also implemented book move openings and minimax searching the whole tree at endgame (due to the small depth). Optimizations such as alpha-beta pruning allow us to check more states during each play. We also search within a time-limit (cases with fewer moves would not search as many states) using iterative deepening and return the best move at a timestamp.

4.2 MCTS

The first version of our Monte Carlo Tree Search agent was a simplified form of a standard MCTS algorithm. When implementing this agent, we tried to create a tree structure so our algorithm could evaluate board states at a depth greater than one. However, this proved insanely difficult, since it had to conform with our interface as well. At the end, this MCTS agent only evaluated at a depth of one, and had no backpropagation that would update its predecessors. In addition, this agent utilized a heuristic that placed a heavy emphasis on corners and edges, with corners being valued at 25 points and edges being valued at 5 points. We also tried to add to this heuristic by taking into account the second outer ring of the game board, since placing a disc in this ring would allow the opponent to take an edge or corner piece. However, this heuristic proved to be perform worse than the heuristic that didn't take into account the second outer ring.

In addition, three agents were implemented with different heuristics. Agent MCTS3 placed equal weight on the corners, edges of the board and more weight on all other board positions. Agent MCTS2 placed most weight on edges, then second most weight on corners and least weight on all other board positions. Agent MCTS1 placed most weight on corners, then second most weight on edges and least weight on all other board positions. To minimize factors that could affect the agent's decision making, we set up a controlled variable where each agent runs 500 simulations or the allocated time divided by 0.1, whichever value is smallest. After completing the round-robin tournament, we observe that out of the three agents, MCTS3 deemed most effective out of the three by having most wins.

4.3 Learning

We designed and trained two different learning agents. The first is a Convolutional Neural Network (CNN) that takes three-dimensional input of size (8,8,2), where the first 8x8 is an Othello board state, and the second 8x8 is the current player, so either 1 or -1. The architecture of this CNN includes three convolutional layers, each with a 3x3 window. We used ReLU activation functions, 2x2 max pooling, and normalization after each convolutional layer. We then added a dense layer and a softmax layer to get a probability distribution over the 64 possible squares of the board, where the probability of a specific square represents the probability of that square being the best next move. We trained this CNN on a dataset adapted from the Kaggle Othello Games dataset mentioned above. To preprocess this data, we turned a single game into a set of all the board states that occurred during that game. This resulted in a dataset of about N=1,500,000. We trained on 80% of this data and used the other 20% as a validation set. We trained with a batch size of 32 for 50 epochs, and were only able to achieve a validation accuracy of 25%.

Our LSTM (Long Short-Term Memory) model was engineered to capture the sequential dynamics inherent in Othello gameplay. With LSTM layers at its core, the model aimed to discern and retain temporal dependencies between successive game states. We approached the problem as a classification task, where the goal was to predict the next game state based on the current one. Employing a categorical cross-entropy loss function, we trained the model using a 75-25 split of the data for training and validation. The output layer, a dense layer with softmax activation, generated a probability distribution over possible moves. Each probability represented the likelihood that the corresponding board configuration would be the resultant next state. Despite training efforts on a subset of the data, the model struggled to surpass a 10% test validation accuracy.

Due to the poor validation accuracies of both of our attempts at learning models, we refrained from integrating them into our tournament due to inadequate performance. This outcome underscores the challenge of accurately predicting optimal next moves in Othello, where heuristic-based strategies often outshine models relying solely on long-term dependencies and datasets of moves made by human players.

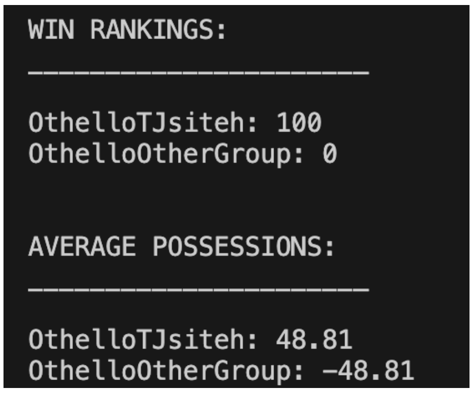
4.4 Results

Agent	Number of Wins
OthelloTJsitef	16
OthelloTJsitea	13
OthelloTJsiteh	13
montecarlo	8
OthelloMCTS3	7
OthelloMCTS1	4
OthelloMCTS2	4
OthelloBasicAgent	4
OthelloRANDOM	3

Table 1: Results from Round Robin Tournament

After preparing all of these different agents, we ran the round robin tournament on the nine various agents to determine which performs best. The nine agents in our final evaluation tournament are shown in Table 1. The top three agents with most wins are the search agents, followed by MCTS, simply greedy search and at last was the agent that would randomly pick every turn.

4.5 Bonus: Competition vs. other Othello group



```
WIN RANKINGS:
--------------------

OthelloTJsiteh: 100
OthelloOtherGroup: 0


AVERAGE POSSESSIONS:
--------------------

OthelloTJsiteh: 48.81
OthelloOtherGroup: -48.81
```

Figure 1: Results from Bonus Competition

After evaluating our own search and learning agents by playing them against each other in a round robin tournament, we collaborated with the other group in class working on Othello to test our top agent against theirs. The agent the other group provided used depth-limited minimax with pruning. To be able to play our top agent against theirs, we integrated their heuristic into our existing Othello interface and GUI. After this integration, a member of the other group signed off on our implementation of their agent's heuristic and logic. We then ran games on our interface, allowing each agent no more than three seconds to decide on the next move. After simulating 100 games between these two agents, with each agent playing 50 games as black and 50 as white, our agent won every game. On average in these games, our agent ended up possessing 48 more pieces than the other group's agent, suggesting a much stronger heuristic. Output from this simulation is shown in the figure below. "OthelloTJsiteh" is one of our strongest search agents.

5 Conclusion

In summary, we attempted to build AI agents to play the game Othello. We created search agents with different heuristic functions, Monte Carlo tree search agents, and a CNN-powered learning agent. All of these agents consistently beat a random Othello agent, and almost all of these agents were able to beat our group members. When we played our agents against each other in a round robin-style tournament, we found that our search agents performed the best compared to other agents. We then played our top agent against the top agent of the other Othello group, where our top performing agent outperformed the other group's agent in all 100 games. If we had more time or chose to continue working on this project, we would continue to improve our learning agents by experimenting with different model architectures and extended training periods. We would also explore different options for labeled Othello datasets that do a better job of measuring the quality of each individual move in a game.

6 Individual Project Contributions

Akshan developed search agents as well as the GUI. He also worked on setting up the round-robin tournament program so we could play agents against each other fairly and in a controlled fashion to get our resulting statistics.

Aknazar worked on the Monte Carlo Tree Search agents and researched various heuristics to implement for the three different agents and also designed a simple greedy search agent.

Jack worked on the CNN agent and integrated the other Othello group's search agent into our Othello interface so that our agents could play against theirs.

Jondash designed, trained, and evaluated the LSTM model. This was done by cleaning, transforming, and feature engineering the kaggle dataset into a suitable format for the LSTM model.

Matt designed the first Monte Carlo Search Agent with the heuristic that placed a heavy emphasis on corners and edges and also designed architecture diagram.

All group members contributed to the project proposal, midterm and final presentations, and the final paper.

References

[1] *Github Repository*, https://github.com/Wild266/verbose-enigma.

[2] Muthukaruppan Annamalai, Vaishnavi Sannidhanam, *An Analysis of Heuristics in Othello* https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf

[3] Hiroki Takizawa, *Othello is Solved*, arXiv preprint arXiv:2310.19387, October 2023.