Genetic Othello

4701 Final Report

Inyoung Hong ih235
David Kahn dmk254

Introduction

In this project, we wished to explore the powers of genetic optimization upon different structures. We originally chose to apply genetic optimization to chess AI, but that was too computationally expensive to have time to evolve the AI to their full potential for examination. So instead, this project applies genetic optimization to different forms of AI for the game of Othello, which is a simpler game. We hoped to see that certain AI structures were more or less conducive to genetic optimization. Of those that performed well, we also hoped to learn new Othello strategies, and understand how the AI structure guided the genetic optimization to such strategies. This project touches upon several AI topics including mini-max trees, genetic optimization, and neural networks. It was written in Java.

The AI we developed were as follows:

1. **Mini-Max Piece-Counter (Flat):** Evaluates the board by the difference between the number of the players' pieces, then applies mini-max.
2. **Mini-Max Weighted Wedge Piece-Counter (Wedge):** Evaluates the board by the difference between the weighted number of the players' pieces, where weights are symmetrically assigned based on position, then applies mini-max.
3. **Mini-Max Weighted Grid Piece-Counter (Grid)**: Evaluates the board by the difference between the weighted number of the players' pieces, where weights are individual to position, then applies mini-max.
4. **Mini-Max Neural Network (MMNet)**: Evaluates the board by a 3-layer neural net, then applies mini-max.
5. **Neural Network (Net)**: Chooses a move with a 3-layer neural net directly.

The questions we asked for this project were:

1. Will a genetically-developed AI be competitive against human players or beginner AI's online?
2. If not already competitive, what level of depth is necessary to make our mini-max based AI's competitive?
3. Which evaluation function is the most effective (Flat, Wedge, Grid, etc.)?
4. How do weight-based AI's compare to a neural net?

5. Does combining a neural net with mini-max lead to better results than a neural net and mini-max alone?

Othello

Othello (also known as reversi) is a two-player game played on an 8x8 grid with two kinds of pieces, denoted black and white. Each each color is assigned to a specific player. To start, the center 2x2 square is arranged with two white pieces on opposite corners, and two black pieces on the remaining corners. Black then goes first.

Each turn, a player must place a piece down that sandwiches (with no unoccupied spaces in between) some positive number of the opponent's pieces between another of the player's pieces, along a horizontal, vertical, or diagonal line. The sandwiched opponent's pieces are then flipped to a piece of the player's color. If no such moves are available, play simply passes back to the opponent.

When neither player can make any moves (such as when the board is filled), the game ends. At that point, the winner is the player with the most pieces of their color on the board.

Because a piece is placed almost every turn, pieces are never removed, and the board starts with 60 spaces open, this makes the game almost always take just over 60 turns. (Chess games can last significantly longer, especially with AI that are bad at closing out games.) Empirically, at an arbitrary point in the midgame, roughly 5-15 moves are available. (This source http://fragrieu.free.fr/SearchingForSolutions.pdf gives a conservative estimate of 10, so this is a fair observation.) This gives an idea of the branching factor of Othello. By considering each possible piece in each possible space, we see a rough upper bound of 3^64 total board configurations, many of which are illegal. (One of the best upper bounds chess has been shown to have is 2^155. See http://tromp.github.io/chess/chess.html. This is on the order of 10^16 times as many compared to that overestimation of Othello's configurations.)

There are two common elements of Othello strategy that develop, aside from obviously counting pieces:

1. The first is to look for stable pieces. These are pieces that the opponent cannot flip at all, or at least cannot easily flip. Stable pieces occur in specific board positions. A corner piece, for instance, cannot be flipped, and nor can any piece of the same color adjacent to such a corner piece. Edge pieces can only be flipped one way (sandwiching along that edge direction) rather than along any of the 4 usual directions (vertical, horizontal, or either diagonal). Obtaining pieces in these stable positions is useful because they are effectually a permanent part of the player's piece count at the end.

2. The second is to maintain a mobility advantage. This is the most important strategic concept of Othello (http://gaurang.org/gothello/guide/guide.html). The idea is for a player to keep the number of moves available to themselves high, and their opponent low. This gives them more tools to deal with situations, while making the opponent predictable. It allows the player to force their opponent to give up the all-important stable corner pieces, or make other disadvantageous plays. The mobility strategy is typically achieved by maintaining a very low number of pieces throughout the midgame, counterintuitively to the goal of maximizing one's piece count. (To see an extreme example of the effectiveness of lowering one's own piece count, see page 7 of https://services.math.duke.edu/~bray/Courses/49s/StudentSurveys/Fall2014/KL_Paper2_GameTheoryOfReversi.pdf. One piece wins against 59).

Experimental Set-Up

For each algorithm scheme, we did the following: Start with a batch of 20 of the same algorithm scheme, each randomly parameterized through an array of numbers (the DNA). Then begin generating new generations iteratively, based on fitness. In each iteration, the whole batch engages in a double round robin tournament, playing once on each side. Win-rate is recorded for white side, black side, and in totality, where each win is with 1 point and each draw worth half. This gives the measure of fitness.The batch then has the members with the lowest points culled, and their spots are repopulated genetically from the better-performing algorithms. This culling and repopulation occurs randomly, biased heavily by win-rate, such that the very top algorithms are definitely not culled, and the very bottom definitely are.

This is accomplished by first setting up a new batch with the best-performing algorithm already included, in a method known as elitism. Then, each of the remaining fitness scores are raised to the fourth power, and 9 are chosen randomly in proportion to that power. This gives higher scoring algorithms a definite advantage for staying in the next round, but does not exclude lower performing algorithms from having a chance.

Then, from the 10 set to remain in the next batch, 10 additional algorithms are generated through sexual reproduction. 2 (possibly identical) algorithms are chosen from the first 10, and each element of their DNA is chosen with a 50% chance to be the element in the same position in their child's DNA. There is then a 20% chance that a given element mutates, with 75% of mutations adjusting the value slightly up or down, and 25% completely randomizing them.

Each algorithm scheme will be run for a comparable amount of time to get a good specimen, and each algorithm scheme will be repeated a few times to account for variance in what the algorithm converges on. For most algorithms, this amounted to running overnight for 300 generations, though Net went quickly so was also given a 1000 generation version.

All but one algorithm scheme (Net) was built upon mini-max, where the DNA parametrized a function that evaluated the goodness of a board. Because Othello is a zero sum game, this further motivates there being no reason to make separate AI for each side in this set up; a good board for a player is exactly as bad for their opponent. The basic mini-max algorithm was built with alpha-beta pruning and set to run to depth 4 for time's sake. It also hardcoded in wins and losses as positive/negative infinite value, because we felt there was little purpose in forcing the AI to learn *to win* in addition to learning *strategy*; winning when possible is clearly the optimal solution. Using Java class inheritance, all but the board-configuration evaluator function could be easily shared between the AI of this type. Setting the depth farther using the same optimized parameters also allowed for increasing the power of these algorithms post-evolution to compare against stronger AI available online.

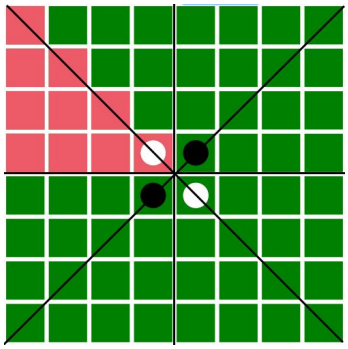
AI Structure Details

Mini-Max Piece-Counter (Flat)

- Evaluation Function: The difference between the number of the players' pieces.
- DNA: No DNA is actually necessary for this AI; it acts a control. However, since the evaluation for Flat is equivalent to the evaluation of Wedge with equal weight on each of the ten squares of the wedge, this was implemented as Wedge with DNA all set to 1.0.

Mini-Max Weighted Wedge Piece-Counter (Wedge)

- Evaluation Function: The difference between the weighted number of the players' pieces, where weights are symmetrically assigned based on position. There are ten squares in each wedge, as illustrated in red on the board below, and these wedges can be flipped symmetrically along the black lines drawn on the board below. Weighting the pieces based on position in the wedge allows for similar positions on the boards (such as each of the four corners) to be weighted the same way.
- DNA: An array of 10 numbers, each representing the weight of one of the ten squares in the wedge.

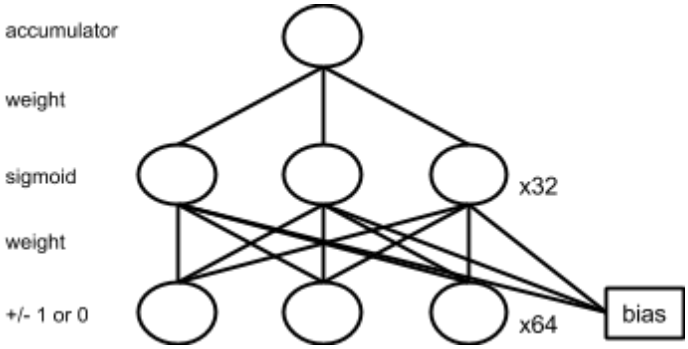


Mini-Max Weighted Grid Piece-Counter (Grid):

- Evaluation Function: The difference between the weighted number of the players' pieces, where weights are individual to position.
- DNA: An array of 64 numbers, each representing the weight of one of the 64 squares of the board.

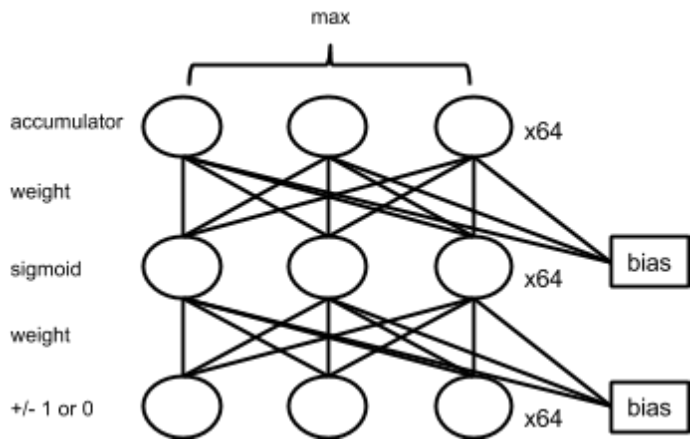
Mini-Max Neural Net (MMNet):

- Evaluation Function: The output of a three-layer neural net. The input layer is of 64 nodes, one for each space, and each node takes on a value of 0 if empty, else positive\negative one depending on the color occupying it.The middle layer is of 32 nodes with a sigmoidal activation function. The output layer is merely a single accumulator node.
- DNA: An array of 66*32 edge weights and biases.



Neural Net (Net)

- Evaluation Function: The maximum legal output of a three-layer neural net. The input layer is of 64 nodes, one for each space, and each node takes on a value of 0 if empty, else positive\negative one depending on the color occupying it.The middle layer is of 64 nodes with a sigmoidal activation function. The output layer is of 64 accumulating nodes, each corresponding to moving in one of the 64 spaces of the board.
- DNA: An array of 2*64*65 edge weights and biases.



Results

The following grids show the evolved weights for the representatives of the Grid and Wedge AI schemes. The other AI schemes, involving neural nets with thousands of weights,, are not nearly so interpretable, so will not be visualized here.

The weightings given by Wedge after 300 generations, where positive weight is good:

69.6							
-1.98	-0.83						
1.38	-5.36	-3.25					
0.60	-7.70	-1.15	-1.67				

The weightings given by Grid after 300 generations, where positive weight is good:

-0.74	6.34	6.66	0.56	4.94	5.14	9.89	11.90
-4.26	-10.9	-9.21	-1.48	-6.50	-9.75	-20.7	-9.08

1.61	-8.81	1.26	-5.25	-4.09	-3.41	-10.5	8.41
1.28	-4.76	-6.21	-1.71	1.50	1.95	-3.69	5.93
6.16	-3.89	6.25	-4.89	-2.7	-7.05	-1.11	1.93
-0.31	-7.79	1.23	0.74	-0.66	0.52	-5.53	2.94
4.43	-2.53	-9.95	-7.64	-4.11	-6.51	-11.9	6.97
8.99	2.62	3.72	1.16	-0.33	5.83	-3.01	17.6

To compare the performance of our AI's, we played each AI strategy against all of the others, first as the black player and then as the white player. The AI strategies based on mini-max (Flat, Wedge, Grid and MMNet) all had the same depth-level of 4 and DNA from the 300th generation of their evolution. In order to get a similar comparison with the neural net AI's, we compared the mini-max AI's with Net (300), which is the 300th generation of Net. However, Net (300) performed quite poorly, so we also included Net (1000), the 1000th generation of Net, in the AI comparison.

We also compared our AI's with online AI's to get a more objective evaluation of our AI's. We played our AI's against the Beginner level on Webversi (http://webversi.com/board.jsp?row=6&col=7) and the Beginner and Medium level on Hewgill (http://hewgill.com/othello/).

The table below shows the results of playing each of our AI's and online AI's against each other. The Hewgill AI only played as white, so there is no row in the table for Hewgill as black.

B = Black win
W = White win
Draw = Draw

		White Player (2nd Player)								
		Flat	Wedge	Grid	MMNet	Net (300)	Net (1000)	Web-versi Beg.	Hewgill Beg.	Hewgill Med.
Black Player (1st Player)	Flat		W	W	W	B	W	W	B	W
	Wedge	B		W	B	B	B	W	B	B
	Grid	B	W		B	B	W	W	B	W

	MMNet	W	W	W		Draw	W	W	W	W
	Net (300)	W	W	W	W		W	W	W	W
	Net (1000)	W	W	W	W	Draw		W	W	W
	Web-versi Beg.	B	W	B	B	B	B		B	B

The table below shows the number of wins and draws for each of the AI structures against each other. The total score for each AI is calculated with the following formula, in which wins are 1 point and draws are ½ point.

Total Score = White Wins + Black Wins + ½(White Draws + Black Draws)

	White Wins (out of 6)	Black Wins (out of 8)	White Draws (out of 6)	Black Draws (out of 8)	Total Score (out of 14)
Flat	3	2	0	0	5
Wedge	6	6	0	0	12
Grid	5	4	0	0	9
MMNet	3	0	0	1	3.5
Net (300)	0	0	2	0	1
Net (1000)	4	0	0	1	4.5
Webversi Beginner	6	7	0	0	13
Hewgill Beginner	3	N/A	0	N/A	N/A
Hewgill Medium	5	N/A	0	N/A	N/A

From this table, we can see that the online AI Webversi performed the best, with Wedge and then Grid being the next best performing AI's. The neural net at 300 generations (Net 300)

performed the worst, but after 1000 generations, the neural net (Net 1000) became better than Flat and MMNet. We can also see that all of the AI's did better on white side than black side.

Only Wedge was able to beat Webversi Beginner or Hewgill Medium at the settings for which we evolved the mini-max AI's (depth 4 search and 300 generations of evolution). This is likely because the online AI make use of opening books and other very specific domain knowledge to aid their strategy. We wanted to compare the effectiveness of the board configuration evaluation functions for our mini-max AI's, so we decided to compare how much farther forward each AI needed to look to secure a win against Webversi beginner.

The table below shows the depth needed in each AI's mini-max search to beat the Webversi AI online. We did not check for depths above 10 because the computation took too long, and depth 10 is just enough to distinguish the power of each. These are indicated with depths >10.

AI Type	Depth to win Webversi Beginner as Black	Depth to win Webversi Beginner as White
Flat	>10	>10
Wedge	6	3 (2 draws)
Grid	7 (6 draws)	6
MMNet	>10 (10 draws)	>10

We also tested each of our AI's and the Webversi Beginner AI against human players, but no one was good enough at Othello to beat any of them. So there is no meaningful test data from human testing, apart from confirmation that even the worst of our AI's were competent and competitive against human players.

Notable Features:

The best performing AI was actually that with the smallest DNA (aside from Flat, the control with no DNA). Wedge, with only 10 slots of DNA, won every time it was on the white side and all but one time it was on the black side when played against all the other AIs we evolved. This suggests that genetic optimization performs best in narrower search spaces. Even Grid did not outperform it, even though it could mimic Wedge by happening to choose symmetric weights, and could also try more options via non-symmetric weights. Upon examination of what makes Wedge so powerful, it was noticed that wedge places very high weight on the corner pieces, lightly positive weight on every edge *except* spaces adjacent to those corners, and negative weight everywhere else. This means Wedge tries very hard to get corners, also wants edges, and avoids spaces that could allow the opponent to get corners, as well as avoiding any interior spaces. During mid-game play, it was also noticed that Wedge typically had around 15 available

moves, whereas its opponent had around 5, because Wedge kept the number of its pieces on the board minimized; this is likely because it tries to get rid of interior pieces. (Note that it doesn't try to lose by getting rid of all its interior pieces early because we hard coded in loss as infinite negative weight.) Thus, it effectively learned to implement the mobility strategy. Attempting to understand how wedge's strategy worked is in fact what prompted looking more into known Othello strategies.

Grid developed a similar strategy as wedge, just not to the same efficacy; simpler turned out better. However, Grid did develop some interesting patterns of its own. The corner spaces of the second-to-outermost layer were given highly negative weight. This is likely because taking them opens up a route for the opponent to get the corners. There is also much more asymmetry in the center of the board, likely encoding a set of opening moves (Wedge's scheme does not have this capability.) However, because the mini-max tree looks multiple moves ahead, the actual openings it favors are not easy to pick out. By testing all 4 possible symmetrically equivalent first moves, it was found able to respond in any of the 3 possible ways, so the opening is not very consistent, at least.

The AI also universally learned to do better on the white side (the player that goes second) despite having been set up with no preference for either side. On a 4x4 and 6x6 board, it is known that white wins with perfect play (see http://ailab.awardspace.com/othello4x4.html for the 4x4 solution and https://web.archive.org/web/20131031041130/http://www.feinst.demon.co.uk/Othello/6x6sol.html for the 6x6 solution), so perhaps white has an inherent advantage on the 8x8 board as well.

For a comparable evolution time (*not* generation number, but total time), the straight neural net (Net) performs better than the one paired with mini-max (MMNet). This suggests that the efficiency of a neural net is not particularly synergistic with mini-max. It would be better to supplant mini-max with a straight neural net than to add a neural net evaluator to a mini-max function. Moreover, merely counting the pieces for mini-max (as in Flat) was better than using a neural net evaluator. Again, simpler is better.

Answers to our Questions

We believe our project to be successful because we were able to answer all of the questions we initially had in pursuing this project.

1. Will a genetically-developed AI be competitive against human players or beginner AI's online?

   Yes. Human players do not really hold a candle to AI's when it comes to Othello. We were initially worried that our genetically-developed AI's would not be able to play

competitively even with many generations, but even 300 generations was completely sufficient to produce AI's that could easily beat humans. We, however, did not come into contact with anyone who plays Othello commonly, let alone competitively, so this may partially explain why humans did so poorly. As for how well our AI's did against other AI's, see the results charts.

2. If not already competitive, what level of depth is necessary to make our mini-max based AI's competitive?

   Some were already competitive, notably Wedge. See the Results charts for specific depth numbers.

3. Which evaluation function is the most effective (Flat, Wedge, Grid, etc.)?

   Wedge. Simpler structures turned out better for genetic algorithms, and Wedge was the simplest. This is an insight into the efficacy of genetic algorithms: they are good for optimizing over narrow search spaces with a lot of built in understanding, and less so over more general spaces. Wedge took advantage of symmetry, mini-max, and hardcoded win/loss evaluations to narrow the search space. Neural nets, however, have a lot more spurious functions they could represent that need to be evolved out. Neural nets have thousands of constants to optimize, whereas weighting has less than 100. Grid needed to optimize 64, and Wedge only 10. See Notable Features for more discussion on how Wedge wins.

4. How do weight-based AI's compare to a neural net?

   They typically do better. As discussed in 3, simpler is better.

5. Does combining a neural net with mini-max lead to better results than a neural net and mini-max alone?

   No. As discussed in Notable Features, they had an anti-synergistic effect when compared based on evolution time (not generation number) to the neural net alone. The increased computational demand of the neural net did not offset any additional power the net provides to mini-max. Some mini-max methods (like Flat) and the neural net were quite comparable to each other otherwise; neural nets are just not good use of computational resources to evaluate for mini-max. When compared based on generation count, however, the neural net with mini-max performs much better than the neural net alone, but worse than other simpler mini-max options. This is likely because there is a lot of power in the mini-max structure to begin with, and because it is (again) more effective to genetically optimize over narrow spaces with few constants than larger spaces with many, and a neural net has thousands of constants to optimize. Looking only at generation count ignores the computational costs of each of these structures, and only

evaluates based upon how many iterations it takes to converge to a good solution; AI schemes that do a lot in an iteration because of lots of computational structure (like mini-max plus a neural net) therefore are unpunished on this metric. Othello is likely a game that is more apt for mini-max to succeed on, being turn based, as opposed to neural approaches, so adding mini-max just makes any approach do better.