Using the Genetic Algorithm to Evolve a Winning Strategy for Othello

Tucker Cunningham
Undergraduate – Computer Science
Stanford University
Stanford, California 94305
tuckerc@cs.stanford.edu

ABSTRACT

This paper is an exploration of strategies in the classic game Othello using the genetic algorithm. By assigning specific values to each position on the board, a computer program can evaluate what move is best in a certain situation. Using the genetic algorithm to discover a good set of values results in the computer being able to play a strategically sound game without using time-consuming game tree searches.

1. Introduction

The game of Othello is a complex board game, mixing elements of go and checkers. A quick summary of the rules is as follows: 1) the board is made up of an 8x8 square grid, with pieces placed one per cell 2) red and white alternate placing a piece on the board, in such a way that one ore more opponent pieces are between the placed piece and another friendly piece. The enemy pieces are then flipped to the placer's side. 3) when neither side can place in this way, the winner is the player with mose pieces on the board. A complete explanation of game rules can be found at the British Othello Federation webside (www.ugateways.com/bof4.html). The size of the branching factor in Othello, combined with how quickly the balance of the game can swing, makes it an interesting and difficult game for computers to play. The standard approach in AI game playing is to make a simple function to evaluate how "well" a player is doing in a certain board configuration, then harness the power of computing to search many moves in the future, and use that knowledge about how the game will play out to choose the most advantageous move. It is necessary for a computer to look many moves in the future because of the inherent difficulty of extrapolating what is a good move by examining only the current board configuration. However, if enough were known about the advantages and disadvantages of holding certain spots on the board, it should be possible to effectively play Othello while only considering one move into the future. This was the philosophy that inspired the application of the GA to Othello.

2. Background

In order to successfully evolve weights for board positions, a strategy for how to assign weights was needed. Although a typical Othello board is 8x8 squares, a 6x6 board will be used for this experiment in order to reduce the complexity of the problem. This should not change how games are played at all, and in fact patterns that emerge in the weighting of this reduced board will probably carry over to an 8x8 board as well. An examination of the Othello board reveals a rotational symmetry that allows us to significantly reduce the complexity of this problem. This rotational symmetry means that there are only really six unique board positions on a 6x6 board. Table 1 shows the 6 base positions in bold, and the rest of the board is filled in to demonstrate how those unique positions are reflected around the board.

Table 1 – Unique board positions and their rotational symmetry

1	2	4	4	2	1
2	***3***	5	5	3	2
4	***5***	***6***	6	5	4
4	5	6	6	5	4
2	3	5	5	3	2
1	2	4	4	2	1

3. Methods

To set up the problem to be solved by the genetic algorithm, several steps needed to be completed. The most important were deciding how to represent the problem in terms that the GA could deal with, namely how to represent the weights as a chromosome, and how to measure the fitness of those chromosomes. Then parameters of the GA run needed to be decided upon, and actual runs made to come upon a solution.

3.1 Representing weights

Once a weighting scheme was arrived upon, all that remained was to set up a GA run to choose the best weights for each of the 6 unique board positions. It was decided to assign each position a value ranging from –2 to 2. The values of this range don't actually matter, simply the ratio between them, which results in the computer "wanting" to hold one position more than the others, so the range was mostly chosen so that the chosen weights would be most meaningful to a person. The only important thing was that the range have both positive and negative values, so that if squares were disadvantageous to own, they could be negatively weighted. This range of values was represented as 6 floating point numbers, each being stored using a binary string of length 16. The GALib software that was used in experimentation has a facility for converting between binary and decimal, so it only needed to be told that the chromosome contained 6 numbers in the range –2 to 2, and how many bits to use to represent them, which essentially decided how precise the values evolved would be.

Because there are 6 weights with $2^{16} = 65{,}000$ possible values, it would seem that the search space is huge. However, there are two factors that reduce this size enough to make the problem solvable with a reasonable population size and number of runs. The most significant of these is the fact that the individual values are not important, simply the ratio between them. This is because the search algorithm which chooses the correct move simply compares scores for different board configurations, all using the same weights, and chooses that which produces the highest score. This means that weights {.2, .3, .6, -.2, -.8, .5} are exactly equivalent to, and will produce the same game behavior, as weights {.4, .6, 1.2, -.4, -1.6, 1.0}. The second factor decreasing the search size is that similar ratios will produce the same behavior as well. If a single square is weighted 5 times higher than another, changing it's weight to 5.1 times higher will not favor it *that* much more. The difference in the games played by these two different weights will be negligible, so they are virtually equivalent. Also, the search space is continuous, there is no absolute correct set of weights, we are simply striving to improve performance. We do not need to arrive at any exact set of weights for the problem to be solved, we simply need a set which performs well, so there is certainly some fuzziness allowed in the correctness of the results. These factors all combine to allow the search space to be tractable even given its significant size.

3.2 Measuring fitness

Once the GA knew how to represent the weights, it needed a way to evaluate the fitness of those weights. This problem constitued the majority of the programming and thought that went into this experiment. A networked game playing client and server were already written from another project, all that needed to be done was to harness this functionality to measure the effectiveness of the given set of weights. This was done by modifying the client program so that if it received a certain flag, it would read a set of weights from a file, and then play a game using those weights.

Each turn, the client would iterate through all of its possible moves, generate the board that would result from each of those moves, and then add up how many pieces both it and the opponent had on that board. It would then move to maximize its pieces in relation to the opponents: essentially greedy search. However, when tabulating the score for a certain board configuration, each piece was not worth 1 point, but was instead worth the weight assigned to the square that it was in. By greedy searching using the weights, the program favored moves which captured highly weighted squares and avoided negatively weighted ones. Once a game were successfully completed, the client would then write what the final result of the game was out to a file.

Once this modified client were written, the fitness function for the weights was simple to write. It simply wrote the current set of weights to be evaluated out to a file, then called the client and server programs to play a game with those weights. Once both of those processed had returned, indicating that the game was over, the fitness function read the results of the game from another file. Raw fitness was assigned based on how badly the weighted program either won or lost the game. This number was later scaled using sigma trunction, which also dealt with the fact that many of the fitness scores were negative.

One interesting problem in measuring the fitness was how to set up the opponent that the weighted program played against. Because the objective of the experiment was to show that brute-force deep searching was not the only way to effectively play, an opponent using exactly this deep-searching strategy was used. The opponent would be configured so that it evaluated boards using a strict greedy search, my pieces versus yours, but looked several moves ahead in the game tree before making a decision. This would mean that the fitness would measure how well a given set of weights peroformed against an opponent looking many moves ahead, facilitating the objective of showing that this is not the only way to effectively play Othello by computer.

3.3 Tuning the fitness measure

Once a basic fitness strategy was arrived upon, it had to be modified in order to arrive at meaningful results. The main variable to be changed was how far ahead to have the opponent program look. Preliminary runs had him only looking ahead 2 or 3 moves. Unfortunately, pure greedy search at this low of a depth is not a great heuristic, and one of the members of the population in the first generation or two would be able to beat it. This is because, although the weights at that early stage essentially resulted in random play, a significant percentage of all random games will at least minimally beat a low-depth greedy search. The starting population size was large enough to include a couple of these, and they ended up looking more fit than they actually were. The only solution to this problem was to have a better adversary, which meant a deeper search depth. Although this did increase computation time significantly, it was the only solution.

Some optimizations were performed to allow searching deeper in the same amount of time, but the most important strategy change was simply being more patient to wait for long runs to complete. It is an interesting commentary on the complexity of Othello that, although moving randomly will not win consistently, it will beat a fairly weak opponent enough times to throw off results. It was necessary to essentially make the opponent much more difficult in order to arrrive at results that would reliably win.

One possible strategy that could have been used to combat the problem of essentially random wins skewing results would have been to play several different games with a set of weights, and then take the average of the fitness score for each game as the actual fitness of that set of weights. Unfortunately, the determinism of the opponent's algorithm meant that playing multiple games that acutally had different results was fairly difficult. Some experimentation was done with introducing non-determinism into the opponent and playing multiple rounds, but being non-deterministic while staying competitive ended up being fairly difficult. Insead, several runs were done, and the top individuals from each run were all

combined to give a final set of weights. By utilizing patterns in many separate, but all highly fit individuals, the possibility of the weights being overfit to a specific game against the greedy opponent was reduced. Perhaps, given more time, exploring the possibility of multiple games to measure fitness could be examined more fully.

3.4 Running the GA

Once a fitness measure and chromosome representation were arrived upon, all that remained was to establish some parameters for a GA run, and then actually do the run.

Population sizes of around 100 were used for most runs. This number was so low mostly to keep runtime down. Because of the factors discussed above that decrease the size of the search space, this ended up being large enough to search through enough of the space to end up with a working result. The number of generations was allowed to vary more between runs, from 30 to 50. Lower numbers were used initially, then some of the later runs were allowed to run longer. This allowed the average fitness to catch up to the fitness of the best couple individuals, so that multiple highly-fit individuals from the final population could be examined.

The standard genetic operators of mutation, reproduction and crossover were used. Mutation was given a typically low probability of 0.1, because there was no reason to set it abnormally high. Crossover was set to 0.8, higher than the standard value of 0.6. This is because the objective of the experiment was to evolve a population that was highly fit and combine several individuals for a final result. By utilizing crossover more than normal, the probability that a single, fit individual simply reproduces and doesn't share its genetic code with the population is reduced. By sharing more genetic code through increased crossovers, it is possible that the best individual may grow slightly slower, but the population as a whole should stay more tightly bunched, resulting in a more useful set of individuals as the end of the run.

The tableau below was used for the final result-producing GA runs. Because of the inability to run multiple fitness tests for each individual discussed above, several runs with slightly different parameters were done, and their results combined to come upon the final set of weights. This negated any false-positives arrived at by testing against a single opponent.

Table 2 – Tableau for GA runs

Objective:	Maximize the score of a game played with weighted board positions, versus an opponent performing greedy search to depth N, where N ranges from 4 . . 6.
Representation scheme:	6 weights, ranging from –2 to 2, represented each by binary strings of length 16.
Fitness:	Fitness is measured by playing a game against an opponent using greedy search. The piece differential of the weighted opponent to the greedy opponent is the measure of fitness, with negative values indicating a loss.
Parameters:	Population size: 100
Termination criteria:	A certain number of generations are run, between 30 and 50, depending on time constraints and opponent search depth.
Result designation:	A combination of the weights of several of the best individuals from a run.

The software used for the GA runs was GALib 2.4.5, available at "lancet.mit.edu/ga/". It provides many advanced GA features, but only the basic algorithm from Goldberg's book was used. One advanced GA feature that did prove useful was sigma trunction scaling, which scores fitness based on deviation from

the average. This is useful because it handles negative raw fitness scores well, which was certainly necessary in the case of this experiment, because any loss to the greedy program resulted in a negative fitness score.

The code used for playing Othello games is derived from the code provided for the CS-221 artificial intelligence class at Stanford, but was modified both to improve performance of the adversary player, and to allow the weighted player to interact with GALib. Many of the performance problems that were encountered during the experiment were due to the interaction between GALib and the Othello codebase. Because the Othello code was a completely separate program that ran separate processes for the server and each of the players, it needed to be launched for every evaluation of an individual during a run. The only means of communication between the GA code and Othello code was reading from and writing to files. Unfortunately, forking processes and file input/ouput are fairly expensive operations, which expended close to half the computation time of each run. Some progress was made in integrating the Othello codebase into the objective fitness function, so that only a single process would be active for the entire run, but complexity and size of the Othello codebase made this an infeasible task given time constraints on the experiment.

4. Results

In every one of the runs performed, an individual was evolved that could beat the greedy opponent by a significant margin. More impressive though, especially the above discussion of the fact that a single game against the greedy opponent is not necessarily an absolute measure of performance, is the fact that the average performance of the population also increases well above 0. This means that by the end of a run, most of the individuals in the population are able to beat a greedy opponent searching far further down the tree than they are. Shown below in table 3 are the abbreviated results of one such run.

Table 3 – Improvement of Population as a Whole

Generation Number	Best of Gen Fitness	Average Fitness
3	2	-24.99
6	8	-16.53
9	18	-9.76
12	28	-4.56
15	28	7.4
18	28	12.16
21	30	15.03
24	30	20.84
27	32	20.50
30	30	21.12

Once we have a satisfactory run that shows a highly evolved population such as this one, we simply take several of the top individuals, by having the GA report its population at the end of the run, and take the average of their weights. Shown in table 4 below are 4 individuals from the final population, and the final weight determined by averaging their weights. The weights are given to only 1 decimal accuracy because this is the format the the GA program output them as, it does not indicate the precision of the actual weight, but we are averaging them anyway, so this should not affect results. Note that especially weight 4 is not completely converged, but that is the point of taking the average as we do. A run to a higher number of generations did not improve the convergence of weight 4, so this possibly indicates that it just isn't that important to overall performance.

Table 4 – Top Individuals and Final Result

Individual	W1	W2	W3	W4	W5	W6
1	1.9	1.9	1.3	1.2	-1.2	0.8
2	1.7	1.8	1.3	1.1	-1.4	0.6
3	1.6	1.9	1.3	-0.5	-1.8	0.5
4	1.8	1.8	1.4	-0.7	-1.3	0.8
avg	1.75	1.85	1.32	0.87	-1.42	0.67

5. Analysis of results

The first thing to notice about the results is that the stated goal was indeed accomplished. With an opponent searching even 6 moves in the future and moving greedily, we are able to beat him without searching ahead at all. The fact that the weights of several of the best individuals all look similar indicates that these are close to the true values of those squares, but they are different enough that it is probably a good idea that we are taking a synthesis of several of the top individuals. Perhaps if we could improve efficiency, doing runs with higher population and number of generations would be able to improve this convergence, especially with the increased crossover discussed above. However, because the obtained weights are so successful, there is no doubt that they are at least approaching optimal.

It is also interesting to try to glean bits of strategy from the values of the weights. It is not surprising that weight 1, the corner weight, is so high because the corner is impossible to capture. Likewise for weight 2, because it is on an edge and more difficult to capture. More interesting is the fact that the values for weight 4 are so different amont equally successful individuals indicates that it is probably not of great importance to the overall outcome of the game. Most interesting is the fact that weight 5 is weighted against so heavily. Perhaps the reason is its proximity to an edge, making it vulnerable to being taken on the way to capturing an edge. This is not true of weight 3 though. This indicates that there is perhaps more to Othello strategy than is immediately obvious, and a fact like this would have been difficult to discover without the problem-ignorant approach of the GA. It seems to have discovered a complexity that is not immediately obvious to an outside observer, which is perhaps the greatest testament of its power.

6. Conclusion

This paper shows that it is possible to successfully apply the genetic algorithm to the problem of generating a heuristic function for playing Othello. Although the 6x6 board operated upon is smaller than an actual Othello board, and the greedy search function that results were measured against is far from the best Othello algorithm available, these are only factors that shortened the time necessary to achieve a positive result. This is essentially a proof-of-concept that genetic principles can be applied to play Othello, and a exploratory foray into how to actually do so. Although the results may be of limited utility on their own, the framework for allowing the GA software to talk to Othello effectively is now in place. Possibilities for further exploitation of this work follow in section 7.

7. Future work

The most obvious avenue to explore in the future is expanding the search space to address the full 8x8 Othello game board instead of the reduced 6x6 version. This was infeasible for this project only due to time constraints. Moving to 8x8 would increase the search space from 6 unique positions to 10, requiring runs with larger populations and more generations to adequately search the whole space. However, this is not the only increase in time that would result. The branching factor would also be significantly higher for the greedy opponent's search tree, meaning much more time would be required for his search. Because this is already the most processor-intensive component of the fitness evaluation, this would increase the time needed for a run significantly. If more time were available, however, these increases in time complexity would be acceptable in exchange for the possibility of developing weights that were applicable to actual games of Othello.

Another avenue to explore in the future would be using genetic programming to evolve an entire evaluation function, not just weights for an established greedy evaluation. One of the major limitations of the weighting scheme is that the weight of each position does not depend on the value of the others. For example, edge pieces are much more valuable if you own all the squares until a corner, because that means that the position is safe from capture. A GP approach would be able to take this into account. Unfortunately, using this strategy would eliminate the rotational symmetry that was so useful in shrinking the size of the search space for the weighting approach. This means that any GP evaluation function would need to take 36 inputs, one for each square on the board. While this is not an intractably large problem, it would certainly require a larger population size, and probably a number of generations in the hundreds to reach any meaningful result. Given many months of processor time in which to perform experiments, this would be the most attractive avenue to follow for future research.