# Computer Systems Research Paper Draft
# Machine Learning with Othello
# Computer Systems Lab 2006-2007

Nick Sidawy, Period 5

June 12, 2007

**Abstract**

Machine learning is a new field that has been coming more into focus over the past years. Having computers being able to adapt to different environments and learn from experiences is important with the progression of technology and the evolution of robotics. However, these deal in complex systems with different factors to consider. In order to learn and apply machine learning it is helpful to start with a more simple system which will facilitate the learning of the basics. To get started, I have chosen the board game of Othello as my system.

# 1 Introduction

## 1.1 Purpose and Goal

The purpose of this research project is to implement machine learning with artificial intelligence for Othello. The reason why I chose this project is two-fold: first, to create a very effective Othello AI for anyone's enjoyment, and second and more academically oriented, is to gain a deeper understanding of machine learning, a subject which I am interested in.

The project itself is broken up into three different uses of machine learning and artificial intelligence. The first is the use of an effective forward-checker for my AI. The second is the use of the genetic algorithm to formulate the best evaluation function for the AI to use. My last implementation is to have the AI learn from each move it makes. Therefore as it plays more and more, it will perform faster and at a higher level. The first two uses are on the beginner side of machine learning, while the last one is more difficult and will be where most of the "learning" will take place.

# 2 Background and review of current literature/research in this area

Machine learning is an extensive field of study. Most of what is done with machine learning is tied to artificial intelligence which is why Othello seemed to be a good vessel for my research. It is simple enough game for me to work on, yet difficult enough to keep me working throughout the quarters. Obviously machine learning is not limited to board games. For example, recently at MIT, students created a small robot with an artificial intelligence. The goal of the project was to implement machine learning so the robot could to teach itself to walk using trial and error. Furthermore, the robot learned to adapt to walking on different terrains, constantly learning from its experiences. The project was a success.

Although this seems as if it is far away from an Othello artificial intelligence, it is closer than one might think. A student last year used machine learning (genetic algorithms) in order to create the best evaluation function. As I have stated, I would like to do something similar to this but also go a step further by having it learn from the its own moves. Most of what I use and learn to complete this project will come from the CSL's Artificial

Intelligence book.

It is important to realize that although my research is being used as gateway to a higher knowledge of machine learning, this type of artificial intelligence is heavily researched and is not strictly beginner's subject. The best example of this is ChessMaster: 10th edition. This is perhaps one of the best board game artificial intelligences and has been able to defeat the best human chess players the world has to offer.

# 3 Procedure and Methodology

## 3.1 Overview, Requirements, and Development plan

The goal of this project is to create an effective Othello Artificial Intelligence

Most of this project will be coded in java because it is fast and contains many of the tools that will be necessary for my project, such as timers and GUI's that are simple to create and use.

The project is broken up into several smaller programs. This way I can work on interconnected parts of the project without having to worry about causing an error if a tweak is made in one program.

The project is seperated into three distinctive interations:

1. The creation of a GUI along with Artificial Intelligence that the computer will use to play games.

2. The use of the Genetic Algorithm in order to find the most effective Evaluation Value set.

3. Having the computer learn from the moves it makes and storing this information for later use in order to increase the level of play and efficiency.

## 3.2 Research Theory and Design Criteria

### 3.2.1 Iteration One

The first component of this iteration was to create an interactive GUI (General User Interface). At first the goal of the GUI was to just be a visualization of the games played, whether they are simulations between two CPU's or games involving human players. However, I realized it would be important

to have a GUI which would be allow the user to play as many games as was thought necessary and would let the user switch between simulations and human played games in one running of the GUI, as opposed to having to end the program and re-run it.

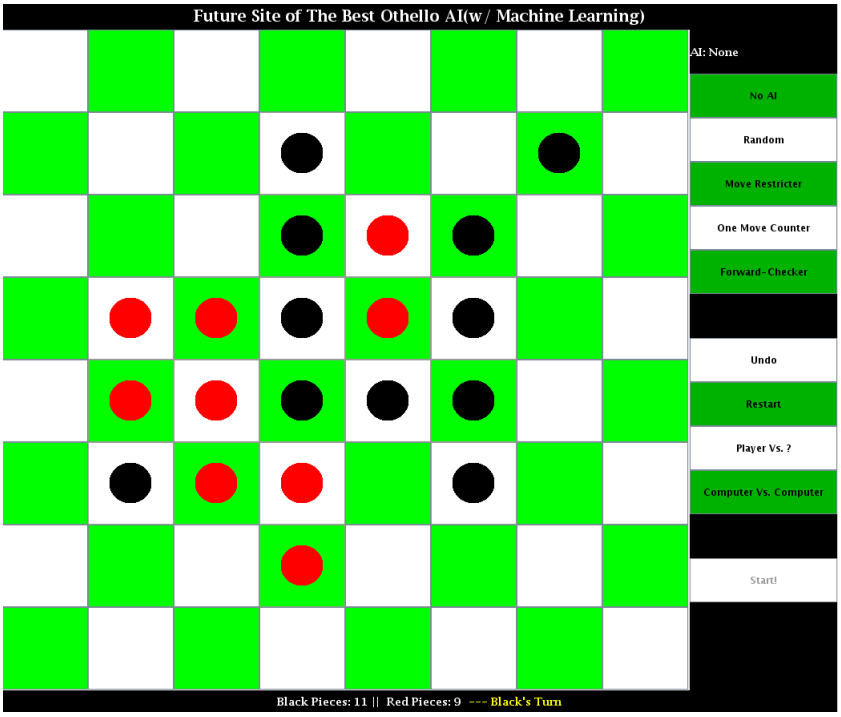


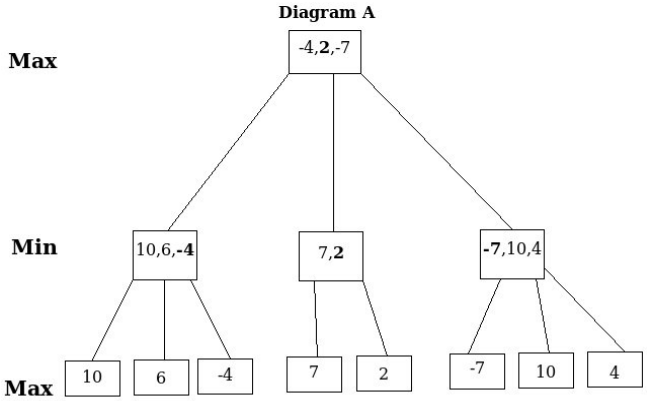

Figure 1: The final version of the GUI.

In order to have an interactive game board which displays the game pieces I had to create a class called "MyButton". The "MyButton" class is an extension of JButton. The important difference between the two is that it contains a paint component which enables the button to be colored anyway necessary. Furthermore, it stores two integers which tell the "MyButton" what color space it is and which piece, if any, currently occupy a given space. Depending on which integers are stored, it will paint itself accordingly.

In the final version of the GUI everything is controlled with the mouse. The user can play or simulate as many games as they wish without having to restart or end the program. Games involving human players include the use of "Undo" and "Restart" buttons along with allowing the human player

to change the type of AI it plays during a given match.

The two main algorithms that were worked on during this quarter are the forward-checking and evaluation methods. The forward-checker works by looking through each possible move that the computer has, making one of the moves, looking at the moves the opponent is presented with, and repeats. In other words, the goal is to traverse a tree of possible moves and to pick the move that will lead to the best scenario down the line. The ply determines how many levels of the tree it goes through. The trick about this algorithm is that at each level of the tree it picks the move that is best for which player it is simulating for. Therefore, the computer assumes that its opponent will play perfectly. This is why it has been labeled the Minimax algorithm. First it looks for the move that is best for it, then at the opponent's move that will be worst for it. (See Diagram A)




This is an example of a three ply Minimax tree. As you can see it traverses three levels and then begins to return evaluation ratings. At the second level it looks at all the evaluation ratings returned from the third level and returns the minimum (shown in bold). At the first level it looks at these selected values and returns the maximum (shown in bold).

The speed of the program is exponentially related to the ply, the amount of moves it searches ahead, of the artificial intelligence ($O(n^p)$ where $p$ is the ply). It is important to realize the substantial toll that making the AI

slightly better has on the runtime.

The other algorithm that was worked on is the evaluation function. This function returns a number to the Forward-Checking method rating how good a specific scenario is for a player. It does this based on the positions of the pieces and amount of available moves for each player. For example, pieces in the corners are very valuable so they will add many more points to the rating then a piece near the center.

### 3.2.2 Iteration Two

The values used to evaluate a given Othello board are very important for picking the best move and I chose to use the Genetic Algorithm to find the best values for evaluating a given board. The Genetic Algorithm follow simple darwinian rules of life. In short, it uses evolution and survival of the fittest to create the

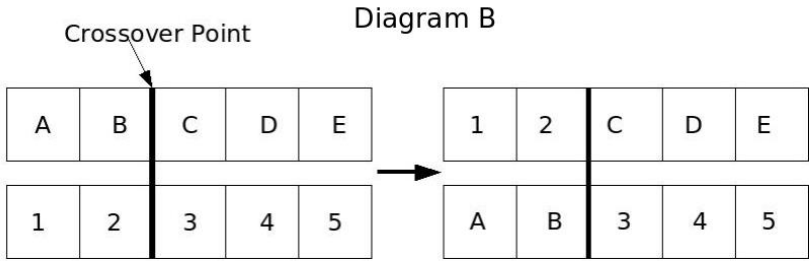
There are three main components of the Genetic Algorithm:

1. The population (A set of different evaluation values)
2. The fitness evaluation (The way of testing how well a set of evaluation values performs)
3. The Splicing and Offspring (The production of a new, improved set of evaluation values)

Each cycle of the Genetic Algorithm has two steps. First, the population (eight sets of evaluation values) is tested against the fitness function. In my problem, I put each evaluation set in a game against an opponent evaluation set which stays constant throught out the cycle (or generation). The two evaluation sets use the same Minimax algorithm in a game of Othello with the only difference being how the two CPU's rate a board. Depending on how well a member of the population does against the fitness function, it receives a score.

The second step involves the creation of the offspring (eight new evaluation sets) through splicing.Thescore a given evaluation set receives from the fitness function determines the likelihood it will be chosen for reproduction. Obviously the better a set performs, the higher likelihood it will be chosen for reproduction. Two sets (Set A and B) are chosen based on the probabilities and a crossover point is chosen at random for the sets. Next, the two sets

will splice by taking all the values before the crossover point of Set A and combining them with the values after the crossover point of Set B. Then, the opposite is performed.




This process is done four times so the offspring (next generation) is created and can be tested against the fitness function. There is also a predetermined chance that a mutation may occur. A mutation is when a value on a specific set is put to a random value and helps prevent the evaluation sets from all reaching the same point (a plateau). In the event a plateau is reached and all the sets in a generation are the same, then the fitness function will become the one of the sets of the generation and eight new sets will be created by random. In theory this will result in the creation of stronger and stronger evaluation sets until an optimal point is reached.

### 3.2.3 Iteration Three

The third iteration is focused more on the machine learning aspect of the project than any of the other iterations. The goal of this iteration was to create a new program to complete three tasks:

1. Storing information from each move the AI makes.
2. Saving the data in a file so it can be used game after game.
3. Loading data from the file and putting it into a HashMap for quick access during a game.

By storing information concerning the move chosen by the forward-checker, it will help the AI be more efficient and effective in later games when similar situations come up. For example, if the AI gets a board that is the same as a

previous board it has a encountered (or a reflection of that board) then all it will need to do is to see what it did the previous time and avoid traversing the tree of possible moves. On the other hand, this information may be used to increase the effectiveness of the AI by using the information after traversing the tree of possible moves in order to search several moves deeper without taking up any extra time. Ideally, a combination of these two techniques would be used.

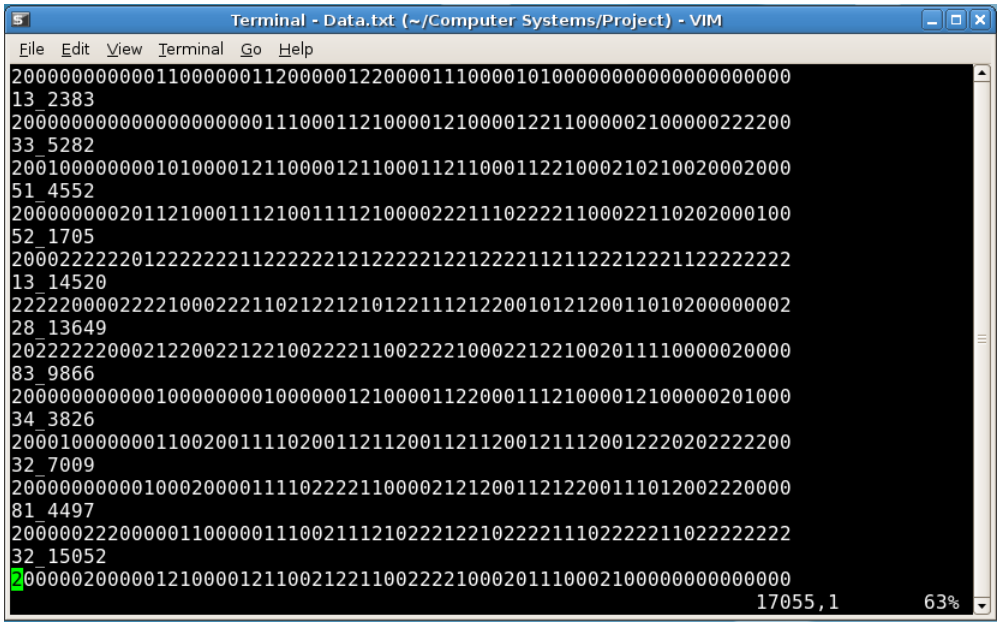



Figure 2: A small portion of the data after it has been saved in a text file.

The first step of the iteration was to decide how to store the information so it could be retrieved quickly and what information to store . The obvious choice for the first question was to use a HashMap. However, the second question was a little trickier. Since a HashMap is being used, the boardstate is the best option as a key. To do this, the boardstate is stored as a string of 0's, 1's, and 2's that is 64 characters long, where 0's empty spots, 1's have a black piece, and 2's have a red piece. At the beginning of each boardstate string a '1' or '2' is inserted to indicate whose turn it is. The value associated with each key is the move that the AI chooses and the evaluation of the board reached with the chosen move. Because an array cannot be stored as a value

in a HashMap these two pieces of data are stored as a single string seperated by an underscore. In order to quadruple the data, anytime data is collected for a board, it is recorded for each of its possible reflections: horizontal, vertical and across the origin.

There are two ways to collect data. Either one can just create all the possible boards and apply the AI to each one. This would obviously take an incredible amount of time. The other way is to just simulate many games between the AI and another opponent and record the moves it makes in those games. I chose to use the second method.

The second step was to save all the data collected from each game into a file, so it could be easily read out by the computer and put back into a HashMap. This is a relatively simple task that is completed by retrieving the keyset from the HashMap, matching each key with its value and storing these two strings one after the other in a text file. This makes loading all of the data simple. The computer just reads in two lines from the file and it has a key and value that goes with it.

The integral part of all this is to modify the forward-checker so that it uses this data to its advantage. At every ply the AI checks the HashMap to see if the current board is stored in the data. If it is not, it continues traversing the tree until the next level where it checks the HashMap again. If it does find that the board is a match, then it will do one of two things depending on how deep into the tree of moves it is. A match found on the first level uses the data and immediately returns the move associated with the board. A match found later on is used to cut out a branch of the tree as if it had already been sifted through and returns the evaluation associated with the board.

As stated before, a combination of efficiency and skill could be increased depending on how the data as used. Using the data to find the move chosen on the first level and returning that move increases efficiency because it cuts out time spent searching through the tree. On the other hand, using the data's evaluation data to traverse the tree allows the AI to, in a sense, search even further than it would normally. This is because each piece of data is derived from searching through a tree of moves five levels. It is important to find a balance between these two options depending on the situation the AI is in.

## 3.3 Testing and Analysis

The first way is to test it against human players. The second way will be to test the AI against previous AI's to see if it has improved or worsened after changes have been made. Margins of victory (or loss) along with running time will determine how well it is performing. Since making the AI better or faster usually requires rather small yet clever changes in the code, it is very easy to work on certain segments from week to week.

The only true way to test for bugs in this sort of AI is to play against it. This is rather frustrating at times because one either has to play enough for a specific situation to arise or recode the program so the situation comes up on the start-up, which is most closely related to "Structural and Functional" testing. In rare instances I am able to use outputs in order to look for bugs, for example, returning the evaluation of the move chosen and making sure that this was indeed the best move to make.

Testing whether the Genetic Algorithm worked involved simply outputing the splicing and making sure everything went smoothly. The more difficult part is making sure that the resulting evaluation sets are truely better than the ones before them. The only real ways to test this is to play against them yourself and determine whether they are more difficult or by playing many trials against a random opponent (an opponent who choose moves randomly) and seeing how well it does on average. As it turned out, the Genetic Algorithm found that the corners were the most heavily weighted factor in determining who was winning in the game, which was expected. What was not expected was how the amount of pieces a player has on a board was weighted just slightly less than the corners. However, I found through testing the AI that the corners still did not have enough influence in the game because that it would routinely give up corner spaces to the opponent. This motivated me to give the corner ratings a higher maximum possibility than the other categories so it would be given more influence in the games. This strategy quickly eliminated the problem.

The final iteration was difficult to actually test because the only way to do so was to have human players play against it. However, after playing enough times I found results that were oppisite of what I had expected. Instead of having the Forward Checker becoming faster and more effective, it became slower and not noticably more effective. Although the running time was not signifcantly lengthened, it was lengthened which means that this method of learning from its moves poses a disadvantage for the AI. The reason for the

increase in running time is most likely due to the use of HashMaps. While the HashMap is generally a quick way to access your data, it becomes slow when it is filled with hundreds of thousands pieces of data. Therefore, when the AI was checking to see if a key was in it at every level, it was, usually, returning nothing which indicates that this was just wasting time. This leads to the second problem. A game of Othello has more than $3^{32}$ possible boards. Therefore, a lot of data needs to be collected if one expects the AI to encounter a familiar board after the first couple of turns.

# 4 Conclusion

My purpose for this research project was to create a strong Othello Artificial Intelligence through the use of machine learning and to gain a deeper understanding of machine learning. The use of the Minimax algorithm has worked very well. It is able to search five moves deep with little to no delay. However, I have discovered that the evaluation values chosen for the Minimax algorithm is more important than how deep the alogorithm can search. This is the main reason why so much time and effort was put into using the Genetic Algorithm. After I had to run the program for extended periods of time, a strong set of evaluation values was produced. Using the Genetic Algorithm has been an interesting experiance. Knowing that all one has to do is run the program and let the computer handle everything itself with no input from the user is a different feeling. I believe that the Artificial Intelligence that I have created up to this point is strong.

The learning part of the project was mostly a falure. It can be assumed that when enough data is collected to cover a significant amount of the possibilities then this strategy would be useful. However, in this situation it turned out that the AI would perform better when this information was excluded from the Forward-Checker.

All in all I felt as if this project was a good experiance. I learned a lot about different forms of Artificial Intelligence and I was able to create an effective Othello AI.

# 5 Acknowledgements

I would like to thank Mr. Latimer for helping throughout my project by providing literature and input. I would also like to thank Lynn Jepsen and David Phillips for testing my program throughout the year.

# References

[1] Hsiung, Sam, and James Matthews, "An Introduction to Genetic Algorithms.", Generation 5. 31 Mar. 2000. 10 Jan. 2007 <http://www.generation5.org/content/2000/ga.asp>.This website gives very concise description of how the Genetic Algorithm works. It goes over the five basic steps involved for each iteration but does not provide any examples. However, it was enough information for me to be able to code the Genetic Algorithm myself and apply to my problem of finding the most advantageous evaluatuion function.

[2] Marczyk,Adam. "Genetic Algorithms and Evolutionary Computation.", The Talk Origins Archive. 23 April. 2004 <http://www.talkorigins.org/faqs/genalg/genalg.html>.This website goes a little further into how the Genetic Algorithm works and explains more of the intricacies of it.

[3] Moreland, Bruce. "Alpha-Beta Search." Bruce Moreland. 4 Nov. 2002. 10 Jan. 2007 <http://www.seanet.com/ brucemo/topics/alphabeta.htm>. This website provided all my information about the Alpha-Beta Pruning method. The website was very good in explaining the process and giving good examples. This information was then used in adding the Alpha-Beta Pruning process to my Minimax algorithm in order to make it run faster.

[4] Russell, Stuart, and Peter Norvig. "Optimal Decisions in Games." Artificial Intelligence: A Modern Approach. Upper Saddle River, New Jersey: Pearson Education, 2003. 163-166. This source explained how the Minimax alogorithm functioned. I used this mostly throughout first quarter to make the actual Artificial Intelligence function for my program, which, in conjunction with an evaluation alogorithm, is able to pick the best possible move based on a give othello board.

[5] Thomson, Elizabeth A. "Teams Build Robots That Walk Like Humans." News Room. 2 Mar. 2005. Massachusetts Inst. of Technology. 23 Jan. 2007 <http://web.mit.edu/newsoffice/2005/robotoddler.html>.