The Use of Genetic Algorithms in Machine Learning; Applications to Othello

Calvin Hayes

April 5, 2006

Contents

1 Introduction

1.1 Problem Statement

The project will be to design a program that is capable of playing the board game Othello at a high level. This program will use a genetic algorithm to optimize weights for specific heuristic values that will decide the machine's line of play.

1.2 Purpose

Genetic algorithms are becoming more and more popular in the field of artificial intelligence as a way to search many different combinations of settings to find an optimal state. The results of the project can be used to determine how successful this particular algorithm is in setting reasonable heuristic weights that can be used by the computer to play Othello better than a program with these weights set arbitrarily. These results can ideally be generalized to other, real-world uses of this algorithm.

1.3 Scope of Study

The eventual goal of this project is for the program to play Othello well. At the most basic level, the machine learning from the genetic algorithm should improve upon any arbitrarily set values for the weights of the heuristics. The program, however, would be a true success if it, using these weights and the heuristics that I have set up, could win games against most or all of the other common Othello programs, those that use a simple 'greedy' minimax search or basic weighted board squares. I would be truly satisfied if the program could beat me. To acheive these results, some knowledge of Othello strategy is necesary, to determine useful heuristics to be weighted. Additionally, other AI techniques, such as minimax trees and alpha-beta pruning will be neccesary to help the program run most effectively and efficiently.

1.4 Background and Review of Current Literature/Research in this Area

1.4.1 Advances in Artificial Intelligence

Usually considered the most successful use of artificial intelligence in game playing is Deep Blue, IBM's chess system that plays at the grandmaster level. Deep Blue used data from many hard-coded games played previously by grandmasters to weight its heuristics, rather than 'learning' the optimal values on its own. Machines have also been 'taught' to play Go, Checkers, and Othello, among other games. Artifical Intelligence is also of unique value in modern video games.

1.4.2 Genetic Algorithms

A genetic algorithm is used to search for an optimal state, composed of an array of individual values used to make decisions. In doing so, it compares the success of each state in acheiving the desired results. Those states that fare the worst are eliminated, and those states that fare the best are morphed together to form new states. Additionally, in a small percentage of cases, these newly formed states are 'mutated' by a small amount by changing one of the values. One can easily see the comparisons to evolution. In effect, this algorithm is simulating the process of Darwinian evolution to build an ideal compilation.

1.4.3 AI and Othello

Many artificial intelligence programs have been created to play Othello. A simpler game than chess, Othello is still copmlex enough to make tactical manuevering soemwhat difficult for a machine. However, over the years, computer programs have improved to become far better than their human counterparts. In August 1997, computer champion Logistello defeated world champion Takeshi Murakami 6 games to none. Logistello was created by Michael Buro and features Prob-Cut search pruning (an improved version of alpha-beta), 17 million training positions, and 1.5 million heuristic weights.

2 The N-Queens Problem

2.1 Development

To use genetic algorithms to solve the n-queens problem, the environment in which the algorithm would run was first created. The board was represented by a list of integers which represented the location of the queens. One queen was always placed in each column, and the data represented the row that she was in for each column. The utility function used for the evaluation of each state of queens was the number of 'attacks' that occured; that is, the sumber of pairs of queens that shared a row, column, or diagonal. Optimally, the vaue of this function is 0. A genetic algorithm was developed with four saved states. Each generation, parents were chosen based on the strength of the utility function for each state and a random component. Of the four states, two became 'parents', one was saved, and one was replaced by the new 'child'. The crossover point for the mixture of data from the parents was set randomly for each generation. The chance of a 'mutation' was set at 20 percent and the chance of a seperate 'introduction' of a new random child was set at 4 percent. Experimentation was done with the option of steadily increasing the values for mutation and introduction, so that the program could be more efficient initially, but have a better chance of positive change once the states were almost optimal.

2.2 Results

The resulting genetic algorithm for solving the N-Queens problem was successful in producing a solution for $N < 16$ relatively quickly (less than 30 seconds or so). It was noted that states very close to the optimal solution were obtained very quickly. This bodes well for the usefulness of the algorithm in real-world screnarios where an exact answer is not needed.

3 Othello

3.1 Development

The Othello environment in which the algorithm would run was taken from the "Othello Referee" that was set up by Torbert in 2004. The environment was tested thoroughly for correctness. Time controls were removed for the

purposes of development, and the function was modified so that the instructions for each move would be taken from within the program as opposed to from imported files.

3.1.1 Heuristics

Basic heuristic valuing functions were written to detect advantages in the following areas:

1. total number of pieces
2. mobility (total number of moves available)
3. corner stability
4. edge stability
5. weighted squares function

Other heuristic functions were written and tested, but only those above were deemed worthy enough to be used in the final algorithm. The heuristic functions were tested for correctness and then basic algorithms that had been built from arbitrary weighting of the heuristics were written and observed. The algorithms all were at least somewhat successful in predicting the outcome of the game from a given board position.

3.1.2 Weighting

The heuristics were combined by making decisions based on a weighted sum of the values returned by each heuristic function. These weights could be set independently and modified at any time. It is the optimization of these weights that is the eventual goal of the genetic algorithm. Each 'player' is actually simply saved in memory as a set of these weights to be used to evaluate a given gameboard using the heuristic functions described above. In order to make the weighting of these functions more straightforward, the values that the heuristic functions could return were translated to values between -1 and 1, where 1 represented the best possible predicted result for the given player and -1 represented the worst possible predicted result. The possible weights were set as integers between -1000 and 1000.

3.1.3 Forward Search

Forward search capabilities were then created using a minimax search with basic alpha-beta pruning. The utility function for the minimax tree was the above mentioned weighted sum of the heuristic functions. Testing concluded that it was beneficial to use forward search techniques in these Othello algorithms, but that the cost was heavy in terms of time. It was therefore determined that it was more efficient not to use forward search in the initial genetic algorithm. Forward search may be used to 'fine-tune' the eventual optimized result, and to test the strength of this resulting 'player'.

3.1.4 The Genetic Algorithm

The genetic algorithm was then constructed. In the current algorithm, 12 'players' are created with random weights for each of the 5 heuristic functions. In each of 100 iterations, each 'player' plays 5 games against a random opponent. The difference in pieces on the board for each game is stored, with positive values representing wins, and negative values representing losses. By dividing the sum of these differences by the number of games any 'player' has played, one can determine the effective piece advantage, per game, of the algorithm compared to a random opponent. It is this 'effectiveness score' that we wish to optimize. After each iteration, the two players with the highest effectiveness score become 'parents' of a new 'player'. Weights for the 'child' are chosen randomly between the two corresponding weights of the parents for each heuristic. Each weight has a chance to be mutated by a small amount. The child replaces the 'player' whose effectiveness score is lowest. As the algorithm runs, the values for the weights of the 'players' should converge to optimal values. 'Players' whose weights are less optimal will be replaced by the 'children' of those 'players' who are best. At the conclusion of the trial, an 'optimized' player is calculated by averaging the weights for each of the top 6 players.

3.1.5 Improvements to the Algorithm

It was conjectured that the heuristic functions had widely varying prediction capabilities in different stages of gameplay. As such, the algorithm was modified to include 20 weights for each 'player'; each of the five heuristic functions has a weight for each of four game stages: opening, early midgame, late midgame, and endgame. When evaluating a board position, each 'player'

used the set of five weights corresponding to the game stage. Additionally, it was determined that effects of early iterations had a continuing effect on the outcome of the algorithm. As such, the genetic component was removed for the first 5 iterations; that is, 25 games were obtained for each initial random 'player before any was removed or created. This was done to prevent poor early results from corrupting the entire process.

3.2 Results

4 Future Study