Applications of Artificial Intelligence and Machine Learning in Othello
TJHSST Computer Systems Lab 2009-2010

Jack Chen

April 7, 2010

Abstract

The purpose of this project is to explore Artificial Intelligence techniques in the board game Othello. The project focuses on the creation and evaluation of AI players for Othello. It explores several techniques used in strong AI players, including improvements to minimax game-tree search algorithms and higher-quality evaluation functions. It also investigates machine learning methods to enable AI players to improve the quality and speed of play automatically based on training and experience.

Keywords: Artificial Intelligence, Machine Learning, Othello

1 Introduction and Background

1.1 Search Algorithms

The primary aspect of most AI players is the search algorithm, which is used to evaluate a board state based on a prediction of future moves from that state. The standard basic game-tree search algorithm is minimax with alpha-beta pruning. I plan to implement several improvements on minimax search.

NegaScout, or Principal Variation Search, is a minimax algorithm that can reduce the number of nodes that must be searched compared to alpha-beta pruning. By assuming that the first variation explored is the best, it produces more cutoffs by searching other variations with a null-window, where alpha and beta are almost equal. If the first variation is not the best, the null-window search fails, and the node is re-searched with a full window. When move ordering is good, NegaScout produces many more cutoffs than alpha-beta pruning.

Another enhanced minimax search algorithm is MTD(f), which uses repeated null-window alpha-beta searches. If the minimax value is outside the null-window, the alpha-beta search fails high or low, which gives a better bound on the minimax value and allows MTD(f) to converge to and find the exact minimax value.

An important way to improve search speed is to cache information about board states that have already been evaluated in a transposition table, which allows the player to avoid repeated searches. Selective search algorithms, such as ProbCut [4] and Multi-ProbCut [5] can further enhance game-tree search by pruning parts of the game tree that probably will not affect the overall minimax value. This allows the player to search much deeper in the relevant parts of the game tree. I will also investigate other search techniques, such as quiescence search.

1.2 Board Evaluation Functions

The board evaluation function is another important aspect of AI players. Traditionally, the evaluation function is based on human knowledge about the game. In Othello, evaluation functions are often based on several "complex" features, such as mobility and stability. However, using a collection of "simple" features, which evaluate patterns in a small number of disks, can improve board evaluation. I will investigate various board evaluation methods such as these.

1.3 Machine Learning

The relative weights of board evaluation features are traditionally hand-tuned. I will explore the use of machine learning to train an evaluation function by automatically optimizing the relative feature weights. There are several machine learning techniques that can be applied to this problem, including gradient descent, genetic algorithms, artificial neural networks, and particle swarm optimization. I will also explore other ways to enable an AI player to improve the quality and speed of play based on experience.

1.4 Other AI enhancements

I will also investigate opening books, which allow much better and faster play in the early game by storing previously computed information about early game board states. I will explore parallelization of the search algorithms, as well as the machine learning algorithms used to train the AI.

2 Development

2.1 Referee

The first part of this project was the implementation of an Othello referee program to run the game. The referee keeps track of the board state and allow two players to play games against each other. The referee supports AI player programs written in multiple languages as well as human players. The referee is implemented in Python and makes use of Python's subprocess module to run each player program in a child process. Running the player programs in a child process allows the referee greater control over the player programs, such as the ability to enforce time limits, and allows them to be implemented in any language. The referee communicates with each player by sending it information about the opponent player's moves and when a game starts and ends.

I have also implemented a graphical user interface for the referee in C++ with the Qt graphics toolkit. The GUI displays the board, animates the players' moves, and allows a human to play easily by clicking on the board. The referee runs the GUI in a child process, like the players, and communicates with it in a similar way. The referee's user interface is implemented in an abstract way that allows it to be easily modified.

2.2 AI Players

The primary focus of the project was on Othello AI players. I have implemented basic AI players in both Python and C++, in order to compare their performance, but most of the AI player development is in C++. First, I wrote players using minimax and minimax with alpha-beta pruning. These are used as a baseline for comparison and for testing improved minimax search algorithms. I have implemented bitboard optimizations, which allow certain board operations, such as finding the legal moves and counting frontier squares, to be performed much faster. I have investigated several AI techniques, including improved minimax search algorithms such as NegaScout and MTD(f).

2.3 Bitboards

I store boards as bitboards using 64-bit integers in which each bit corresponds to one of the 64 squares on the board. Each bit indicates whether or not a piece of a certain color is on that square. One of the integers represents black's pieces and another represents white's pieces. The use of bitboards allows great speed improvements in certain operations, such as finding all possible moves or counting frontier squares, with bit manipulation techniques. The bitboard optimizations made the overall speed of the AI about 5 times faster, enough to search about one ply deeper. Bitboards are also advantageous in terms of memory use, since they are very compact.

2.4 Evaluation Function

I have explored Machine Learning techniques for the optimization of evaluation function parameters. I created a program to generate Othello boards for this training. This program repeatedly plays two AIs with randomized move selection against each other. The players have a small probability of choosing suboptimal moves in order to generate a variety of boards.

My evaluation function takes a linear combination of a set of features, including the number of pieces on different types of squares (in particular, on corners and squares adjacent to corners), the number of possible moves, frontier squares, and parity. The weights for these features are trained using a batch gradient descent method. The game is divided into one stage for each of the possible number of pieces on the board. A separate set of weights is trained for each stage, starting from the last stage. The target value of each board for the stage is determined with a minimax search based on the already trained weights for later stages. The weights are then converged to minimized error with a batch gradient descent algorithm. In each epoch, a modified line search is used to determine the distance the weights are modified in the direction of steepest decent. This method repeatedly multiplies the learning rate by two and checks whether this set of weights reduces the error. This dynamic learning rate helps avoid problems with a fixed learning rate:

extremely slow convergence when the learning rate is too small or convergence to a poor set of weights when the learning rate is too large.

3 Expected Results

I will evaluate Othello AI players by playing games between various AIs. Overall, the success of a player would be judged based on the scores in the games and the number of games won. I will implement multiple players using different algorithms and techniques. I will then compare their success to evaluate the effectiveness of these algorithms. Players using machine learning would be expected to improve as they play more games. I expect the project to result in a fairly strong Othello AI player.

I have chosen to work with AI game-playing because games like Othello are highly constrained while sufficiently difficult to allow significant exploration of AI techniques. Although the AI player implemented in this project will be designed for Othello, many of the AI approaches are game-independent, and could even be applied to similar problems other than games.

References

[1] J. Baxter, A. Tridgell, and L. Weaver. "Experiments in parameter learning using temporal differences."
http://cs.anu.edu.au/~Lex.Weaver/pub_sem/publications/ICCA-98_equiv.ps

[2] K. De Jong and A. Schultz. "Using Experience-Based Learning in Game Playing"
http://www.tjhsst.edu/~rlatimer/papers/gina.pdf

[3] D. Carmel and S. Markovitch. "Learning Models of Opponent's Strategy in Game Playing"

[4] M. Buro. "ProbCut: An Effective Selective Extension of the Alpha-Beta Algorithm"

[5] M. Buro. "Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello"

[6] M. Buro. “Toward Opening Book Learning”