A PARTIAL ANALYSIS OF OTHELLO

By

ROD KEVIN MCABEE

Bachelor of Science in
Mathematics
Oklahoma State University
Stillwater, Oklahoma
1978

Bachelor of Science in
Computer Science
Oklahoma State University
Stillwater, Oklahoma
1982

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1994

OKLAHOMA STATE UNIVERSITY

A PARTIAL ANALYSIS OF OTHELLO

Thesis Approved:

J Chandler

Thesis Advisor

[illegible]

Blayne E. Mayfield

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I offer special thanks to Dr. John Chandler, my principal advisor, for his friendship, insight, assistance, and willingness to have frequent beneficial scholastic discussions concerning this project. Without his drive and support, I would not have completed this thesis. I also wish to thank the time spent by Dr. Hedrick during my sojourn in his department and the support given by Dr. Mayfield who replaced Dr. Hedrick on my committee. I also wish to extend thanks to Dr. K. M. George for his support.

I wish to thank Mark Vasoll and Roland Stolfa for providing a stable computing environment. I am grateful to Oklahoma State University for the fiscal support given to full-time employees. I appreciate the extension given to me by the Graduate College to complete this project.

I wish to thank my wife, Peggy, and my son, Kevin for their understanding and love during this project.

Finally, I wish to thank God for his patience and mercy. May He be praised forever, Amen.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Othello, the Game

Othello is a two-player game using an eight-by-eight square board similar in size to a chess board. The initial board position is shown in Figure 1. Players take alternate turns. One player is represented by the black pieces while the other player is represented by white.

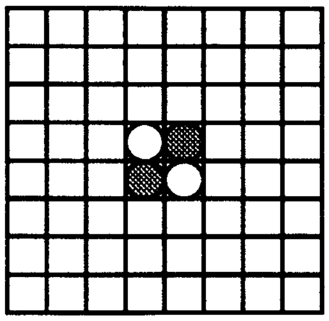


Figure 1

The object of the game is to have the most pieces at the end of the game. The pieces are black on one side and white on the other. A piece may be played if the player can position his piece on an empty square such that it results in bracketing a continuous line of enemy pieces between the new piece played and another piece whose color is the same as the new piece. This line may either be a horizontal, vertical or diagonal line. All bracketed enemy pieces are flipped, becoming the same color as those that bracketed them. In fact, the playing of one piece can result in more than one such line. If no piece can be bracketed, the current player must pass. The game ends when either all 64 squares have been occupied or both players must pass. Unlike chess, Black has first move. Therefore, from the initial board setup of Figure 1, the number of possible moves for Black's first move is given by Figure 2.

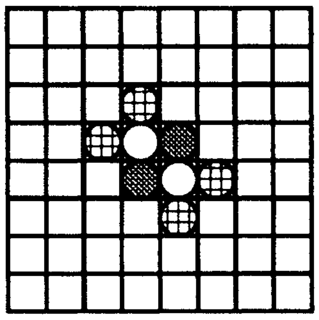


Figure 2

At this point in the game, it is immaterial which one of the four possible squares is played due to symmetry. Thus arbitrarily selecting an empty square and positioning a black piece yields Figure 3.

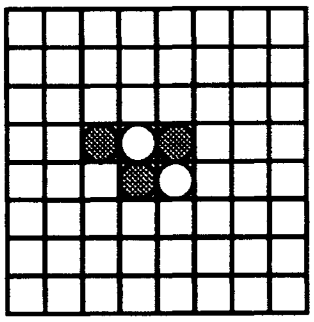


Figure 3

Since a white piece has been bracketed between the newly positioned black piece and another black piece on the board, that white piece is flipped which yields Figure 4, the end of the first turn. It is now White's turn to move. White has three possible moves, and the game would continue from this board position.

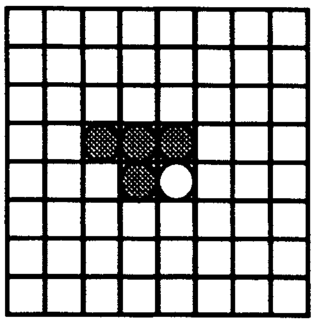


Figure 4

Game Playing

Game playing on computers has utilized a variety of methods in an attempt to play at the level of human play. One of the approaches still in use today was described by C. Shannon [13] in 1950 and dealt with playing chess. John von Neumann [14] had previously shown that a strategy based upon present board position and future possible moves could theoretically be determined for games of perfect information such as chess.

Essentially, the game is described by a given set of rules that determine the transition from one state or board position to the next or rather, the next possible group of alternatives. This series of transitions can be viewed in the form of a game tree as seen in Figure 5 for Othello at the start of a new game. For example, as shown previously, Black has four moves at the start of the game. It was also claimed that it was immaterial which move was made due to symmetry. After Black moves, White has three choices of moves. Next, Black will have either three, four, or five possible moves depending upon the move made by White.

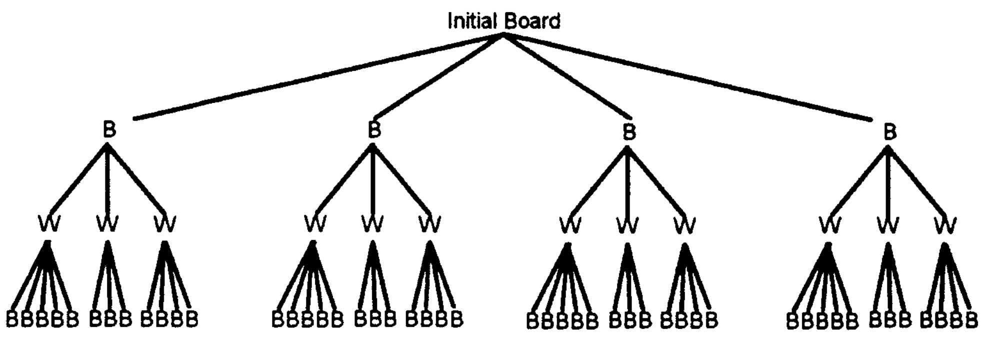


Figure 5

The transition from Black to White or White to Black is referred to as one ply (one level in the tree). The maximum ply depth of the game tree for Othello is sixty. The amount of time required to search the entire tree is extremely large and is covered later. Therefore, due to time limitations, the search depth is limited. This still results in a large number of alternative board positions at the bottom of the game tree called leaf nodes.

An evaluation function is defined to compute various aspects of a given board state in terms of piece count, stability, and so forth. This evaluation function is applied to the leaf nodes of the game tree resulting in a numerical value reflecting the approximate 'goodness' of a given board position.

The next step is to attempt to determine which position will more likely be reached. If the leaf nodes represent a move for Black, then it is in Black's best interest to select that child node of a given parent node whose value is superior. A parent node is any node that has children. Child nodes who share a common immediate parent are siblings. Those values for Black are then "backed-up" to the parent node. One value exists per node. At this level in the game tree, it is now White's turn. White would best be served by picking the smallest value or minimum for the backed-up Black values. These minimums are then backed-up to the appropriate parent of the previous level. Since it is Black's turn, Black will once again select the best or maximum value. This process is repeated and the appropriate minimum or maximum values are backed-up until the current board position is reached. At the current board position, that move associated with the maximum value, based upon the evaluation function, should be the next move. This process is referred to as the Minimax algorithm. If the entire game tree is not or cannot be searched, then the game may be unaware of a better move (as judged by the approximate evaluation function) that would have been revealed had a deeper search been made. This is referred to as the horizon effect.

Other algorithms exist which perform the same function as the Minimax algorithm, with the same results, but have superior speed because they prune portions of the game tree based upon information discovered as the game tree is searched. These algorithms always choose the same move as the Minimax algorithm; they just do it faster. Two of these algorithms are branch-and-bound and alpha-beta pruning. These algorithms will be discussed later.

CHAPTER II

BACKGROUND

Samuel's Method

Samuel [10] describes some methods he utilized in machine learning he had applied to the game of checkers. One of the methods used an evaluation function whose coefficients and respective signs were continually modified during play. He used alpha-beta pruning to search the game tree.

The terms of the evaluation function consisted of coefficients multiplied by various parameters associated with aspects of the game such as mobility, advancement to a king, board position and so forth. These parameters are generally counters whose values increase in relation to the number of times that the condition is satisfied. The coefficients were powers of 2. He identified 38 parameters, but only utilized 16 at any given time during execution of the program, due to limited computer memory. This was due to the computer resources available to him at that time. His program swapped out parameters based on certain criteria. This swapping introduced some complexities.

When the program played itself, one side (BETA) used an evaluation polynomial that was not modified during play while the other side (ALPHA) adjusted its coefficients. Should ALPHA win the game, BETA is given ALPHA's values for the coefficients. Should ALPHA prove inferior, the leading term's coefficient was set to zero. A baseline for the program's level of play was established by having the program play a book game with its learning process disabled. This could then be compared to those moves recommended in the book game. In addition to the program's level of play, a method is needed for measuring changes made in the evaluation polynomial. The only way the program can measure improvement is by the scoring polynomial, which is continually modified. Therefore, Samuel [10] measured this improvement by the following method:

- *Compute the value of the evaluation polynomial and save this value at each step.*
- *Compute the backed-up score for all board positions to a given depth.*
- *Compare the initial board score, as saved from the previous move, with the backed-up score for the current position. This difference between the scores is called "Delta".*
- *Maintain a record of the correlation existing between the signs of the individual term contributions in the initial scoring polynomial and the sign of Delta.*
- *After each play, an adjustment is made in the values of the correlation coefficients, due account being taken of the number of times that each particular term has been used and has a non-zero value.*
- *It is necessary to recompute the scoring polynomial for a given initial board position*

after a move has been determined and after the indicated corrections in the scoring polynomial have been made, and to save this score for future comparisons, rather than to save the score used to determine the move.

The current author does not completely understand the entire process that Samuel used. This is due in part to ambiguity in language and insufficient information in the article.

Figure 6 summarizes the current author's understanding of the basic principles behind the method. This summary does not include the details of how the changes were actually computed, but is an overview of the process. A ply depth of 'M' is assumed for the calculations involving either move 'K' or the state of the board 'X.' An additional assumption that Delta is large enough to warrant modifying the evaluation function is also made. The current author has concluded that where Samuel [10] says:

...computing the scoring polynomial for each board position encountered in actual play and by saving this polynomial in its entirety...

and

At each play by Alpha the initial board score, as saved from the previous Alpha move...

are referring to the same value. 'E_k' is intended to mean the evaluation function in effect at the beginning of the 'k-th' move in the game prior to changing the evaluation function. The initial board score for the 'k-2' move is understood by the current author to be computed at move 'k' using '$E_k(X_{k-2})$' as the evaluation function. The reason that Samuel [10] recomputes the initial board score is to correct a defect involving Delta. His program may even recalculate the board position again based on given values of Delta. However, it is not clear to the current author whether those changes or modifications include recalculating all intervening board positions, or not.

Side	Move	Actual Play (IBS)	Backed up Score	Delta	Recompute after changes - Replaces IBS
A	K-2	$E_k(X_{k-2})$	$E_{k-2}(X_{k+3})$	$E_{k-2}(X_{k-4}) - E_{k-2}(X_{k+3})$	$E_{k+2}(X_{k-2})$
B	K-1	$E_k(X_{k-1})$	NA	NA	NA
A	K	$E_{k+2}(X_k)$	$E_k(X_{k+m})$	$E_k(X_{k-2}) - E_k(X_{k+m})$	$E_{k+4}(X_k)$
B	K+1	$E_{k+2}(X_{k+1})$	NA	NA	NA
A	K+2	$E_{k+4}(X_{k+2})$	$E_{k+2}(X_{k+2+m})$	$E_{k+2}(X_k) - E_{k+2}(X_{k+2+m})$	$E_{k+6}(X_{k+2})$
B	K+3	$E_{k+4}(X_{k+3})$	NA	NA	NA
A	K+4	$E_{k+6}(X_{k+4})$	$E_{k+4}(X_{k+4+m})$	$E_{k+4}(X_{k+2}) - E_{k+4}(X_{k+4+m})$	$E_{k+8}(X_{k+4})$

Figure 6

Magg's Use of Values for Squares

Maggs [7] uses two arrays containing values for the squares. One array uses values at the start of the game. The second array is used for the endgame. The only other change is modification of the values associated with the squares next to the corner squares when the corner square has been occupied. He states:

> *Undoubtedly it could be improved by introducing a number of other changes reflecting particular board configurations and the possibility that a square might have different values for Black and White in some circumstances.*

Frey's Observations of Values for Squares

Professor Frey [2] made several observations concerning game strategy. Of specific interest is the comparison of play between one program whose strategy was to play whatever piece flipped the largest number of pieces in a turn, against another program that played the square that had the highest value. The latter strategy proved to be more effective. Each square was assigned a priority number. Once again, no other basis for determining the value of the squares was given. The values apparently were assigned based on common sense and maybe trial-and-error.

Lee and Mahajan's Criticism of Samuel

Lee and Mahajan [6] make no reference to values assigned to specific squares, but make several references to table lookups and pattern matching. However, they do address the method that they utilize for learning. They state that:

> *Learning schemes such as Samuel's signature table algorithm must learn a large number of parameters. In order to control the number of parameters, quantization is often necessary. Unfortunately, this quantization results in the 'blemish' effect (Berliner [1]):*
>
> > *a very small change in the value of some feature could produce a substantial change in the value of the function. When the program has the ability to manipulate such a feature, it will frequently do so to its own detriment.*

The signature table algorithm was another method that Samuel investigated after examining the self-modifying evaluation function. Lee and Mahajan[6] recommend the use of Bayesian learning, claiming that:

> *While other learning programs learn to differentiate good features from poor ones or to imitate expert's moves, Bayesian learning learns the optimal concept, namely, "moves that lead to a win."*

This project does not investigate Bayesian learning, but this reference was included to introduce the potential for future work.

Game Tree Search Methods

In addition to Samuel's method and other individuals' work in the game of Othello, how the game tree is searched is another important aspect in the determination of the values for the squares. This is important due to the time required to analyze all possible opportunities of play in the game. Specifically, a maximum bound on the number of positions in the game can be approximated by considering the following analysis.

There are sixty-four squares; a bit string consisting of sixty-four bits is used to represent whether or not a square is white, and another sixty-four bit string used for black. A one in the first bit string indicates a white piece while a zero indicates a non-white state, non-white because the square could either be empty or black. A one in the second bit string indicates a black piece while a zero indicates a non-black state. If the same bit location in both bit strings is zero, then the square indicated by that location is empty. Specifically, or both bit strings, then take the complement. In the resulting bit string, a one indicates the square is empty. Based on this, the following bit string represents the initial condition of the board at the start of the game for black:

```
0000000000000000000000000001000000001000000000000000000000000000
```

or in board form

```
00000000
00000000
00000000
00001000
00010000
00000000
00000000
00000000
```

And the next bit string represents the initial condition of the board at the start of the game for white:

```
0000000000000000000000000000100000000100000000000000000000000000
```

or

```
00000000
00000000
00000000
00001000
00010000
00000000
00000000
00000000
```

When placed together, the initial state of the board can be viewed as:

```
0000000000000000000000000001000000001000000000000000000000000000
0000000000000000000000000000100000000100000000000000000000000000
```

Remember that a square may have one of three states: empty, white or black. A square cannot be both black and white at the same time. Therefore, based on the representation used, zeros can exist in the same bit location in both bit strings, but ones cannot. This fact complicates the analysis. For example, certain bit patterns cannot exist. Both strings cannot consist of all ones. In fact the total of all ones never exceeds sixty four. Some unanswered questions that resulted from this project are: "How many bit patterns cannot exist?" "Do these bit patterns have a pattern or can they be determined?"

If one ignored concerns associated with data structures, then an upper bound could be determined based upon permutations. Since a square can have 3 states and 64 squares exist, 3^64 is all possible permutations, excluding the possibility of permutations which cannot exist. The proof that some permutations cannot exist will be done by showing one example. It is impossible to reach the board position such that a black piece exists in any or all corners for the second move. A combination exists to represent this unreachable position. Therefore, more patterns exist than do possible positions.

During a given run of sixty four games, the program kept track of the maximum number of possible moves at any given turn for any given game. Based on the data generated by the program, the following product is an empirical result of an upper bound:

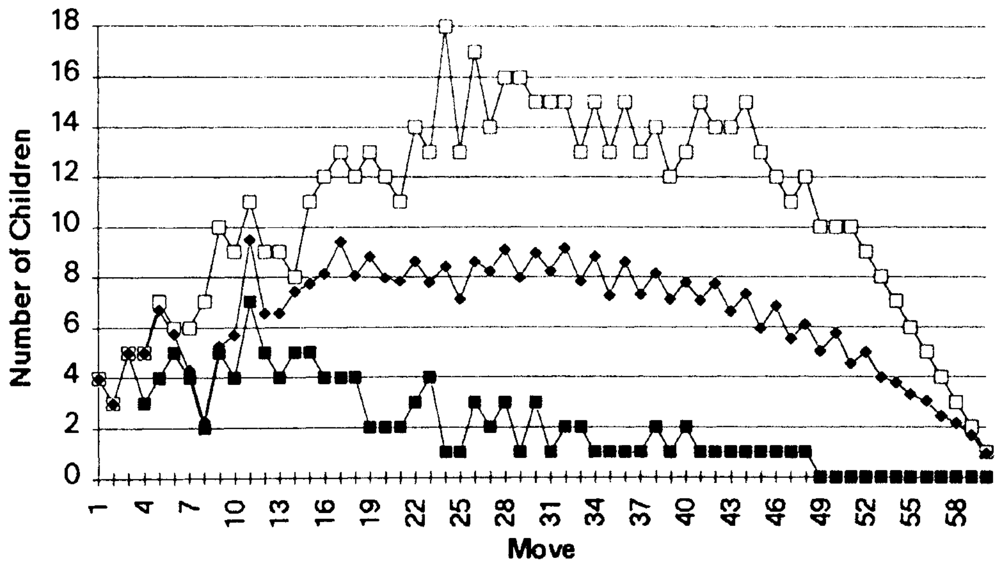


Figure 7

The lower line in Figure 7 with solid squares represents the minimum number of branches during a sixty game run of the program. The top line with empty squares represents the maximum number of branches at any given move. The middle line with the solid diamonds represents the average number of branches for any given move of the sixty-four games.

Minimax

The basic concept of the Minimax algorithm was presented previously in the Introduction. The algorithm, with minor modifications, is described by Horowitz and Sahni[5] as:

```
int Minimax( depth , board_position)
{       ans = -infinity;
        if (depth == 0) return( evaluation(board_position) );
        if (num_of_children == 0) return( evaluation(board_position) );
        for (kid=1 ; kid <= num_of_children ; kid++ ) {
                ans = max (ans, -Minimax( depth-1 , board_position) );
        }
        return(ans);
}
```

Branch-and-bound, Alpha-Beta

Algorithms for branch-and-bound and alpha-beta from Horowitz and Sahni[5] with minor modifications are:

```
branch_and_bound(depth,board_position,beta)
{       if (depth == 0) return( evaluation( board_position ));
        if (num_of_children == 0) return( evaluation( board_position ));
        ans= -infinity;
        for (kid=1;kid<num_of_children+1;kid++) {
                ans= max( ans, - branch_bound(depth-1,board_position,-ans));
                if (ans >= beta) return(ans);
        }
        return(ans);
}
```

```
alpha_beta(depth,board_position,alpha,beta)
{
        if (depth == 0) return( evaluation( board_position ));
        if (num_of_children == 0) return( evaluation( board_position ));
        ans=alpha;
        for (kid=1;kid<num_of_children+1;kid++) {
                ans= max( ans, -alpha_beta(depth-1,board_position,-beta,-ans));
                if (ans >= beta) return(ans);
        }
        return(ans);
}
```

Knuth and Moore [3] provide a detailed analysis of branch-and-bound and alpha-beta pruning in their paper. As can be seen in the two algorithms, they are very similar. The difference exists in passing the lower bound as a parameter in the alpha-beta algorithm while setting its value to minus infinity in branch-and-bound. As a variable, it changes value based upon exploration of the game tree.

Knuth and Moore [3] state that branch-and-bound examines the same nodes as alpha-beta "until the fourth level of look-ahead is reached.... On levels 4,5,..., however, procedure F2 (alpha-beta) is occasionally able to make "deep cutoffs" which F1(branch-and-bound) is incapable of finding."

Figure 8 contains a modified excerpt from two of the graphs in Knuth's paper. The tree demonstrates the returned values for the branch-and-bound algorithm and the alpha-beta algorithm. The numbers associated with the terminal nodes indicate the value of the evaluation function. At each branch in the tree, the backed-up value for branch-and-bound is indicated first if different values exist for the alpha-beta algorithm, otherwise only one value is shown if branch-and-bound is identical to alpha-beta. In the example shown, branch-and-bound finds the same cutoffs as alpha-beta except for the circled 7. Alpha-beta finds that cutoff whereas brand-and-bound did not.

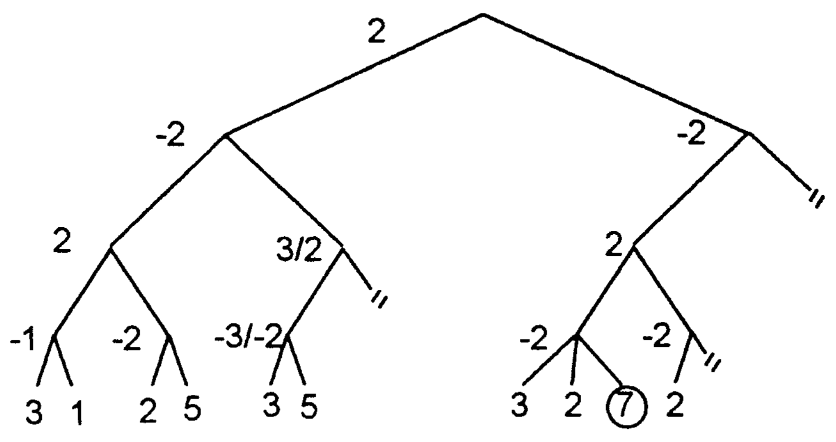


Figure 8

CHAPTER III

PROBLEM DESCRIPTION

Values Determined as Coefficients in the Evaluation Function

Several programs exist for playing Othello; however, all publications reviewed to date indicate that any initial values assigned to squares on the board are estimates. This project in its early stage investigated initial values assigned to the squares using the basic approach in Samuel's method [10]. As a natural development, the project investigates the assignment of varying values to the squares throughout the game. Restating, if the coefficients of an evaluation polynomial are associated with values for squares on an Othello board, this project investigates whether or not the values for these coefficients stabilize under Samuel's method.

Values Determined by Successful Selection

At any given point in the game, a player has either zero or more possible legal moves. A second method of determining square values will be based on increasing the value of the square selected from the group of possible legal moves. The square selected will increase in value compared to those squares not selected. Ideally, a search with a ply depth equal to one would indicate the same move as a search of depth greater than one as the evaluation function is modified over time. At any given move, a square's value is determined by averaging the value of the square that was established in the previous game with the returned value of the evaluation function for a search depth of one ply and increasing the value of the square indicated by the alpha-beta search of a depth greater than one by a value proportional to the number of possible moves. If the square selected by the search of a depth greater than one matched the square selected by a search depth equal to one, no modifications are made to the values of any of the squares.

Conceptual Problems Encountered in Samuel's Method

A basic problem in using the evaluation function is identifying the appropriate parameters. It was and still is unclear how to determine if you have enough parameters to adequately model your problem. Similarly, how do you know if magnitude of any of your parameters get too large such that the value of the parameters is biasing the evaluation function too much in one direction? In this case, the value of the parameters starts to perform the function of the coefficients; it is unclear whether this is good or bad. For example, a piece count is used to provide direction toward a win. If the value of the piece count is very small compared to the total value of the evaluation function, then it no longer provides significant direction. If the piece count value is an order of magnitude larger than the evaluation function, then it will completely override other parameters that might need to have more precedence at that point in the game.

The next question was how to initialize the coefficients. Samuel had his program set up to use or discard parameters under various situations. As a result of this, he had a mechanism for initializing the coefficients as if they had been in use for some time. The current author chose to have the exponents of the coefficients initially set to zero which makes the value of the coefficients equal to one. No negative values were set. The current author believes that some of the positive coefficients could be translated to negative coefficients by shifting the initial range such that some are negative and some positive. Moreover, since some of the parameters where counting is 'number of good minus number of bad', then if the 'bad' states outnumbered the 'good' states, that term would in fact be negative.

Since Samuel modifies the evaluation function for the previous turn based upon the Delta computed by the backed-up function, how do you really know that a possible poor position wasn't the result of a move prior to that one used to compute Delta? Regardless, having made a decision on the approach to use, should all intermediary changes to the coefficients be redone or left alone?

Samuel used the Delta calculation with the understanding that a negative difference implied that the evaluation function was in error and should be modified by decreasing those coefficients that had greater weight and increasing those coefficients that had lesser weight. He then somehow calculated some correlation coefficients to guide the amount of change to the actual coefficients of the evaluation function. This correlation coefficient was based on terms between a backed-up evaluation function and the prior turn of ALPHA. The current author has some concern that the coefficients should have different values at different points in the game. Therefore, how can you change the values based on correlation coefficients computed between the backed-up evaluation function to the value of the evaluation function for a search depth equal to one? What if the coefficients should have different values at different points in the game? Samuel addressed this issue by having four different evaluation functions utilized at different times in the game.

Dr. Mayfield posed two questions during the formal proposal. "Since you are modifying the evaluation function, how do you know that your game is trying to win? How do you know that the values you determine using this method are any good?" Although not asked, also implied was, "How do you know when you have the best values? How do you know when to stop?"

CHAPTER IV

METHODS UTILIZED

Grouping Squares

The basic approach utilizes the fundamental concept described by Samuel [10] in order to investigate initial values for the squares. The board consists of sixty-four squares, however only sixty pieces can be played as the four initial pieces are set up at the start of the game. The first step is to evaluate whether the values under investigation need to be determined for all sixty squares. If a subset can be used, determine the size of the subset and which squares comprise the subset.

Additional lines have been drawn to help provide a reference for the symmetry of the initial board in Figure 9.

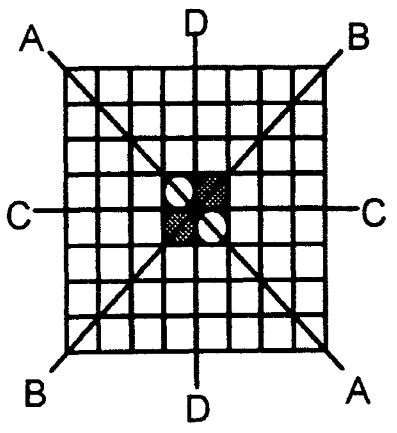


Figure 9

If the board is folded (or rotated) on axis AA or axis BB, it can be seen that symmetry is maintained as one black square lay on top of the other. However, folding the board about axis CC or DD does not maintain symmetry. In this case, a white square now coincides with a black square. Rotating the board 90 degrees clockwise or counter-clockwise does not maintain symmetry as the white squares are where the black positions used to be and black squares are on white positions. Rotating the board 180 degrees in either direction does maintain symmetry. This results in the categorizing the squares of the board in the following groups as indicated in Figure 10. In Figure 10, the two squares marked by a 1 are grouped together; the four squares marked by a 2 are grouped together.

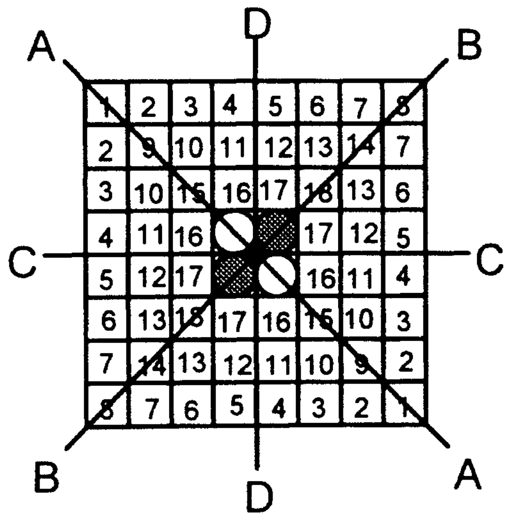


Figure 10

The sixty squares can thus be grouped such that only the values for the shaded subgroup in Figure 11 will be examined. The subgroup consists of eighteen squares. This identifies the squares whose values we will attempt to define via Samuel's method.

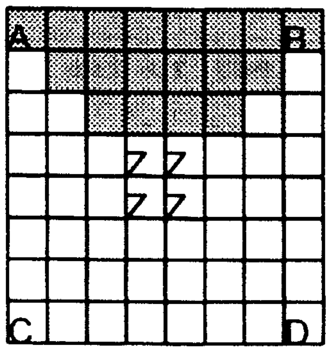


Figure 11

Repeating, squares A and D can be represented by A, and squares B and C can be represented by B. Squares indicated by Z cannot be played as they contain the initial pieces at the start of the game. Therefore, the project investigates the values for the squares using this classification or grouping of the squares. However, this classification could be modified in future work so as to compare the values of all squares except the Z squares in terms of corresponding to the values of the coefficients for the terms.

Since the values for the squares are being determining also by successful selection during play, sixty values will exist for this approach.

Adapting Samuel's Method

The very basics behind Samuel's method are to compare the backed-up value of the evaluation function with the value of the prior move by ALPHA. Based on this comparison, make changes to the evaluation function. The current project compares the backed-up value of the evaluation function and associated recommended move with the values of the evaluation function of all of the immediate children to the current board position. In the best of all possible worlds, both the current value and the backed-up value of the evaluation function should indicate placement of the same piece. If that occurred, no changes were made to the evaluation function. If the current value of the evaluation function indicated playing a different piece, then a term by term difference between the current values for the recommended move and the current max value of the evaluation function was computed. In other words, the terms of the current maximum value were subtracted from the terms of the current recommended move. The term with the largest positive difference had its coefficient reduced by a factor of two while the term with the largest negative difference had its coefficient increased by a factor of two. This moves the evaluation function towards recommending the placement of the same piece as the backed-up value.

This addressed the author's concern in that the coefficients can now have different values at different points during the game. If the evaluation function at depth equal to one starts making the same recommendation as the function backed-up from depth equal to k over time, then the game gives the appearance of learning. However, this does not address the issue of whether the evaluation function adequately models the game.

Search Algorithms

The alpha-beta search algorithm was utilized for most of the project. The Minimax algorithm was used to verify the functionality of the branch-and-bound algorithm and the alpha-beta algorithm. How the algorithms were modified so that they could be utilized in the program is described in the section dealing with the program.

Values Determined by Successful Selection

It should be noted that the values determined by move selection are not used in the evaluation function. That could result in a tendency for the values to either continually increase or decrease. For example, assume a given square was selected last time and its value increased. If that value is also used in the evaluation function, then it will increase the tendency to probably select that square again versus some other square whose value did not increase. Therefore, the values determined by move selection are only used for determining the order of the search in the alpha-beta algorithm. These values are also grouped by move. For example, there is no way that the corner squares can be played in the first move. Therefore the value of the corner square will not be evaluated in the first move. Over many games, the pieces are grouped in terms of whether or not a square was played in a given move. This does not necessarily correspond with the theoretical possibilities, as the entire game tree was not searched. By the same token, the value for a particular square may then vary based upon the move under consideration. In general, the value of the square played is bumped up for a given move, while the values for the other squares that could have been played, but were not selected are bumped down. The amount of adjustment is currently arbitrary, but proportional to the number of pieces vying to be played and the value of the evaluation function for a search of ply depth equal to one. For example, if it is possible to play one of eight choices, then the selected move beat out seven contenders. The amount of adjustment probably should be refined.

CHAPTER V

OVERVIEW OF PROGRAM

Fundamental Data Structures

The program utilizes two arrays whose size is sixty four, the same as the board size. One array represents the white pieces on the board while the other represents the black pieces. Since they are linear arrays, a formula was created that calculates the adjacent squares. Other arrays are used to track the values of the coefficients, parameters, and so forth. In general, all arrays are either one-by-sixty-four or two-by-sixty-four. The exception are those arrays containing values for the parameters and coefficients.

Organization of the Program

The program consists of the standard functions to initialize the appropriate items with one driver function that handles input and alternates between players. Several functions where created to address checking for legal moves, playing a piece, searching the game tree, displaying the board. The only real thing of interest is the use of Samuel's method as the program follows the typical organization of most board games.

Implementation of Samuel's Basic Algorithm

The current author implemented a variation of Samuel's method. The approach is fairly straight-forward. The program compares the backed-up value resulting from the alpha-beta search of depth=K with the values resulting from a search of depth=1. If the maximum value of the depth=1 search corresponds to the same piece indicated by the depth=k search, no modification is made. If the maximum value of the depth=1 search indicates another piece, the difference between each term of the maximum and the recommended piece is made. This difference is computed on the evaluations made at the depth=1, not a difference between the maximum depth=1 and the depth=k values. This allows for the value of the coefficients to be independent at different moves throughout the game. In general, large positive differences have their corresponding coefficients reduced while large negative differences have their corresponding coefficients increased.

Implementation of the Search Algorithm for the Game Tree

The alpha-beta search as given by Figure 12 is indicated by the bold lines; the non-bold lines are those added to the algorithm for the purposes of the project.

```
alpha_beta(depth,cur_color,alpha,beta)
{
   calculate_number_of_children();
   if (depth==0) return(evaluation(cur_color));
   if (num_of_children==0) return(evaluation(cur_color));
   ans=alpha;
   for (kid=1;kid<num_of_children+1;kid++) {
        save_current_board();
        make_move(Child[kid],cur_color);
        if (cur_color == WHITE) pass_color=BLACK;
        else                               pass_color=WHITE;
        temp_ans=-alpha_beta(depth-1,pass_color,-beta,-ans);
        if (ans < temp_ans) {
           ans=temp_ans;
           if (DEPTH==depth) square_to_move=Child[kid];
        }
        restore_board();
        if (ans >= beta) return(ans);
   }
   return(ans);
}
```

Figure 12

In order to make use of the alpha-beta algorithm, additional code must be written to track which was the actual move that corresponded to the value returned, along with actually generating the tree, modifying and restoring the board positions, etc. The computed values of the squares are utilized to order the search.

Evaluation Function

The parameters used in the evaluation function are listed:

- invulnerability
- number of potential directions that can flip a square

- number of directions that actually flip a square
- number of potential pieces that can flip a square
- number of actual pieces that can flip a square
- square stability based upon occupancy of edge squares
    - horizontal direction
    - vertical direction
    - both diagonal directions
- piece count
- number of empty adjacent squares
- number of adjacent squares occupied by enemy
- eighteen parameters associated with occupancy of a square that has a corresponding coefficient to use in Samuel's method

Figure 13 represents the number of directions that potentially can flip a given square. For example, the corner squares are invulnerable. They can not be flipped from any direction. Therefore, their value is zero. The other edge squares between the corner squares can be flipped in one of two directions if piece position allows. All other squares have the potential to be flipped in any of eight directions. The number of directions that can potentially flip a given square will never exceed the numbers given, but in the course of the game, the number of directions may and will diminish as the game progresses. Therefore, while the corner squares are always invulnerable, other squares will become invulnerable as the game progresses.

0	2	2	2	2	2	2	0
2	8	8	8	8	8	8	2
2	8	8	8	8	8	8	2
2	8	8	8	8	8	8	2
2	8	8	8	8	8	8	2
2	8	8	8	8	8	8	2
2	8	8	8	8	8	8	2
0	2	2	2	2	2	2	0

Figure 13

Figure 14 demonstrates another parameter that indicates the potential maximum number of pieces which may flip a given square. This parameter, like the preceding parameter is of more value earlier in the game rather than later as the actual positioning of the pieces decreases the value of both parameters. However, what is not evident is when either of these parameters should be given less weight.

0	7	7	7	7	7	7	0
7	11	23	23	23	23	11	7
7	23	27	25	25	27	23	7
7	23	25	27	27	25	23	7
7	23	25	27	27	25	23	7
7	23	27	25	25	27	23	7
7	11	23	23	23	23	11	7
0	7	7	7	7	7	7	0

Figure 14

While two other parameters, which represent actual values instead of theoretical, correspond to these potential parameters, the estimated computational cost of tracking actual values as the game progresses was believed to be too high and so these parameters are mentioned but not used. It should be noted that parameters might have been utilized if the author's program was rewritten and made more use of tables to minimize the computational cost. An attempt to offset this was made. The program does track the number of empty squares which have the potential to flip each color. This merely corresponds as to whether or not any given square is a legal move. The program also maintains a piece count by color for each move. This is necessary to provide the game a direction, as the evaluation function is continually modified or at least has the potential of being modified. Another parameter counts the number of directions that each empty square has the ability to flip the enemy, and the number of directions for the enemy. Again, the actual number of pieces flipped is not tracked due to computational effort. If the number of pieces flipped were to be tracked, then the program has merely completed almost everything needed for one more ply evaluation. So the trade-off for some of the parameters not used was based on the time needed for evaluation. If evaluation of said parameters corresponds to the time needed for an additional ply calculation, it was not used.

The number of invulnerable squares occupied by color is another parameter.

Another measure of stability is ownership of edge squares. In Figure 15, the edge squares associated with a particular internal square are shown. This square has edge squares that correspond with horizontal, vertical, and diagonal lines drawn through it. Parameters are associated with the group of horizontal lines, the group of vertical lines, and groups of diagonal lines. The diagonal groups are associated with top and bottom, left and right diagonals in order to break the board into sections.

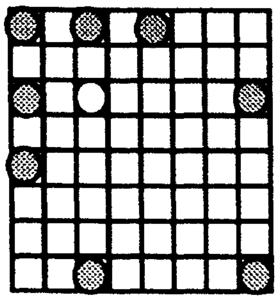


Figure 15

And last, but not least, there are parameters associated with the group of squares described by Figure 11.

CHAPTER VI

ANALYSIS OF RESULTS

Analysis of Values

During the course of the project, the program was run with and without ordering the squares for the alpha-beta search. The search time utilizing the order based on the value of the squares is approximately twenty to twenty-five percent less than not using any specific ordering. In the current author's opinion, that is an empirical indicator that the values of the squares determined by selection have value.

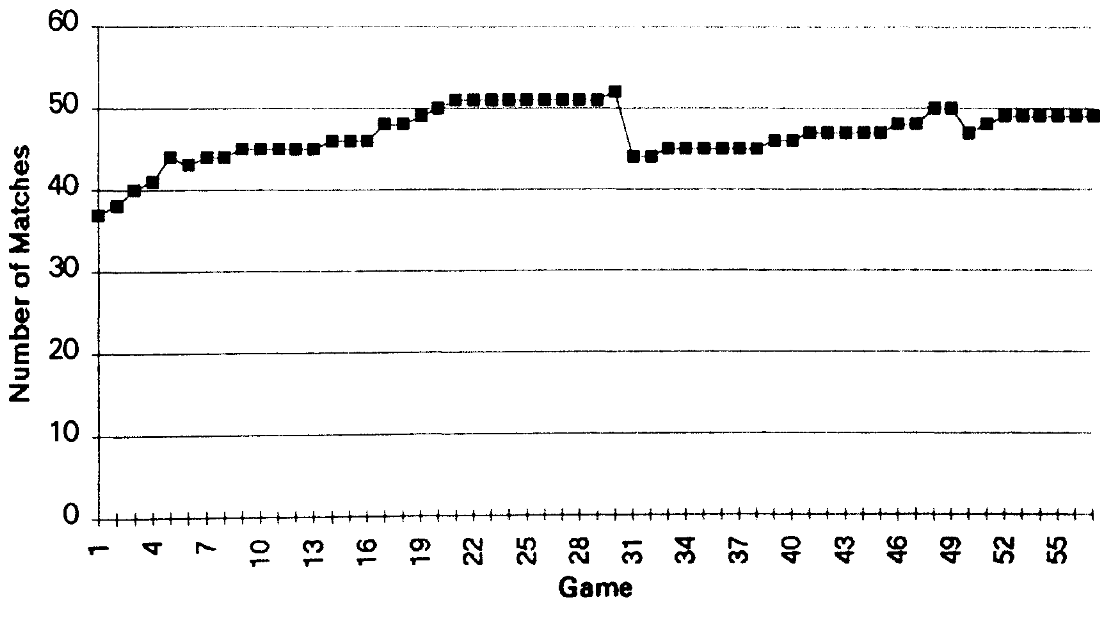


Figure 16

Figure 16 shows the number of times the backed-up value recommended the same piece to be moved as did a depth=1 search. Therefore, on this run, the program started its search over seventy-eight percent of the time with the move that the backed-up value would recommend. However, this is not proof, in the author's opinion, that the values are the best values that can be achieved. Other runs with various ply depths had different values, but the tendency to improve the number of matches still occurred. One of these runs only matched twenty-one times at the beginning, but was matching in the high thirties toward the end of the run.

The square values based on the values of coefficients corresponding to the squares is demonstrated in Table 1. Each one of the entries represents the value of the exponent per move after a run of sixty-four games. The presence of large isolated positive and negative values indicates a lack of stability of the results.

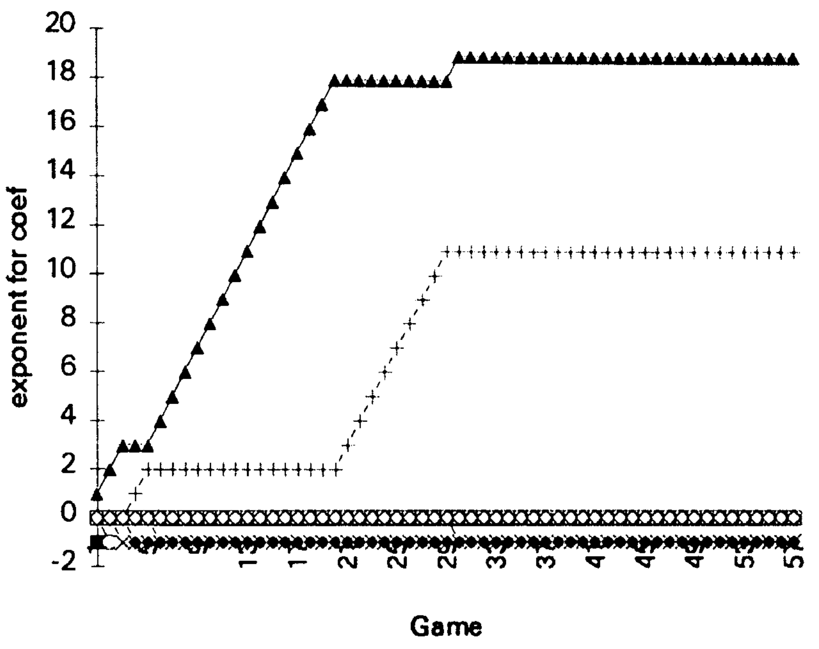
Exponents for the Coefficients

Move	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	-2	0	0	0	0	0	0	0	0	0	0	0	3	0	2	-1	2	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	-1
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	-9	0	0	0	0	0	0	0	0	7	0	-1	6	0	0	0	-1	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0
11	7	0	5	0	0	0	0	0	0	-1	-1	0	-1	0	-1	-1	-1	-1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	-1	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	1	9	-1	0	0	-1	0	0	0	0	1	-1	-1	0	0	0	-1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	1	0	0	0	0	0	-1	0	0	1	-1	0	-1	-1	0	-2	-1	-1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	-8	-1	0	0	-1	0	-1	0	5	-1	0	-1	-1	-1	0	-1	0	0
20	-3	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	-1	0	0
21	-6	0	0	0	-1	0	0	0	-1	0	-1	-1	9	0	10	-1	-2	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	-6	10	0	0	-1	2	0	0	-1	4	-1	0	0	0	0	0	-2	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	10	-1	0	1	2	0	-1	2	-1	-1	-1	0	-1	-2	-1
26	-11	0	0	0	0	0	-1	-1	0	6	0	0	-1	0	8	5	0	0
27	-1	0	0	0	-1	-1	0	0	0	0	2	0	-1	-1	0	1	-1	-1
28	-14	0	0	20	0	0	0	0	0	0	0	-1	-1	-1	0	-1	-1	-1
29	-16	-1	-1	0	5	-1	0	0	0	-1	-1	2	8	-1	-1	-2	0	0
30	-10	-1	0	0	-1	0	0	0	-1	0	5	12	-1	0	-1	0	-1	0
31	-11	0	0	0	3	2	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	0
32	-6	-1	0	0	4	-1	0	5	0	-1	0	-1	0	0	0	-1	4	0
33	-3	0	0	0	0	-1	0	0	0	-1	-1	-1	0	-1	-1	0	-2	0
34	-4	-1	0	0	7	4	-1	0	0	-1	0	0	-1	-1	0	0	0	-1
35	-7	0	0	0	8	0	0	0	-1	-1	-1	-1	-1	0	0	0	0	0
36	-6	0	0	0	2	0	0	0	0	3	-1	-1	0	0	0	1	-1	3
37	-10	0	0	8	0	5	0	0	-1	-1	-1	-1	-1	-1	0	-2	-1	-1
38	-6	-1	0	0	0	0	0	0	0	-1	2	0	0	-1	0	-1	-1	-1
39	-8	0	-1	-1	0	0	12	-1	0	-1	-1	-1	-1	0	-1	-2	-1	-1
40	-8	-1	0	0	0	0	0	0	0	-1	8	-1	5	0	1	-1	-1	0
41	-8	0	12	0	0	0	2	-1	0	-1	-1	0	0	4	-1	-1	-1	-1
42	-23	-1	0	-1	-1	2	0	-1	0	0	0	-1	9	0	0	0	-1	0
43	-2	0	1	0	0	0	0	0	0	0	0	2	-1	0	5	0	-2	0
44	-8	-1	0	0	0	0	0	-1	0	-1	1	0	0	-1	5	0	0	0
45	-1	0	-1	0	0	-1	4	0	-1	-1	5	0	-1	0	-1	-1	-1	-1
46	0	0	0	0	0	-1	-1	-1	0	0	-1	9	0	0	-1	0	0	0
47	3	0	0	0	0	0	1	0	0	-1	3	-1	0	1	0	-1	-1	-1
48	-12	-1	0	-1	0	0	-1	0	-1	-1	10	2	-1	0	-1	12	0	0
49	-7	0	-1	-1	4	1	1	0	1	-1	-1	-1	-1	0	-1	-1	-1	-1
50	-5	-1	8	0	-1	-1	-1	0	2	-1	10	-1	-1	0	-1	0	4	-1
51	-1	0	-1	1	0	-1	0	0	6	0	3	-1	-1	1	0	-1	-1	-1
52	-9	0	0	0	-1	-1	-1	-1	0	0	-1	4	-1	-1	4	1	2	0
53	-7	1	-1	0	12	0	0	0	0	-1	-1	-1	-1	0	-1	0	-1	-1
54	0	-1	0	-1	0	1	-1	0	1	0	-1	0	0	0	0	8	0	0
55	-5	3	-1	-1	8	0	1	0	0	0	-1	-1	-1	0	0	-1	-1	-1
56	-16	-1	-1	0	0	-1	-1	-1	-1	-1	13	3	9	-1	2	-2	-1	-1
57	-1	0	0	-1	0	-1	1	0	0	-1	-1	-1	0	0	0	-1	0	0
58	-18	-1	0	0	-1	0	0	-1	-1	4	-1	-1	-1	-1	-1	-1	0	0
59	0	-1	0	-1	-1	0	0	-1	5	0	0	-1	2	0	-1	-1	-1	0
60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1

The graph in Figure 17 shows the value of all eighteen coefficients for the fifty-fifth move over a run of games. The first coefficient (bottom graph) was separated from the remainder in order to make the graphs scale better with the plotting package that was used. The first coefficient for this series of games had a value of zero in the first game, but changed from zero in the fifth game to minus one.



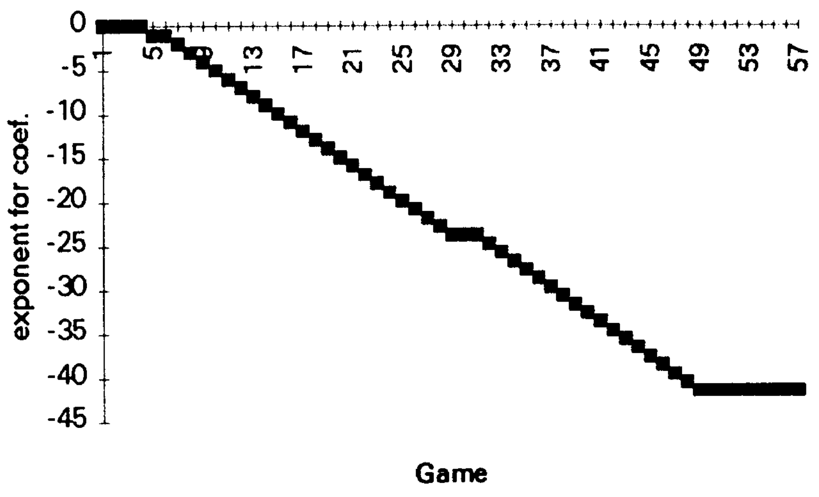


Figure 17

Comparison of Search Methods

The Minimax, branch-and-bound, and alpha-beta algorithms are compared in Table 2 for ply depths of 1, 2, 3, 4, and 5, respectively. The game timings for the methods were calculated after zeroing out all data files. In effect, the program was started each time with no acquired changes for every run. That allows for one-to-one comparison. The other option would be to make several (at least thirty) runs for every instance to even pretend to have some statistical validity.

Depth of Ply	Minimax	Branch-and-Bound	Alpha-Beta
1	11.19 sec	11.18 sec	11.18 sec
2	30.63 sec	21.65 sec	21.66 sec
3	351.64 sec	125.25 sec	125.27 sec
4	1242.65 sec	479.68 sec	461.84 sec
5	36973.37 sec	3659.41 sec	3558.79 sec

Table 2

CHAPTER VII

SUMMARY, RECOMMENDATIONS

Summary of Project

The purpose of this study is to analyze values for the squares of Othello utilizing a method developed by A.L. Samuel. The project's intent was not necessarily to write an Othello program, however an Othello program written by the current author was utilized as a vehicle for Samuel's method. The method utilizes the alpha-beta algorithm for searching the game tree, and modifies the coefficients of the evaluation function at each move. Some of the coefficients of the terms of the evaluation function represent the values of the squares. At the same time, values for the squares were determined based on move selection.

The method produced results that were empirically shown to have value. This was accomplished by using the values to order the search for the alpha-beta algorithm. A twenty to twenty-five percent reduction in search time resulted from use of the ordering. The values determined by move selection change over time and are a crude form of learning. For example, the current author has a tendency to play the same game opening. If a different opening is used, the programs values as determined by selection would shift over time.

The program works towards a win as it had a direction based on piece count by color whose respective coefficients were not modified during play and whose magnitude was predetermined by the current author to provide sufficient weight. If the sign of the piece count was reversed, the program demonstrated a clear tendency to learn to lose.

Values were viewed on a per move basis, as some squares could not be played until the game had progressed. Therefore, sixty sets of values were used.

The values of the coefficients stabilized over time. However, using different search depths could change the values dramatically as the horizon is pushed down additional plys. The game had a tendency to repeat games until random play at the start of the game was added.

The results in Table 1 clearly show some instability; isolated large values are not reasonable. The approach used to analyze values appears to function as hoped as the coefficients seem to have a tendency to stabilize on any given move, but the values are not necessarily smooth from move to move. Therefore, it is the current author's opinion that this project has not determined final values for the squares. Moreover, any final values that would be ultimately determined would have to be taken in context if applied to another's project.

Future Work

The program learns to play at different levels of play depending upon the number of plys searched in its learning mode. This horizon effect has been noticed by the current author when he plays the program. Due to the time required to have the game learn at ply depths greater than five, only a few runs were made during the project's lifetime at depths of six and seven. It is the current author's opinion that a ply depth of nine to twelve or more plys is needed for the method to be highly effective. The number of other parameters used to establish direction must then be minimized. This level of search was determined based upon the number of moves needed to reveal the impact of occupying the corner square without indicating its relative importance in the early stages of the game.

In addition to increasing the depth of the search, an analysis of the differences between Delta determined from a search of depth k and k+m might be of interest.

Another question is how large Delta should be before a change is made, and where should the change be made? For example, assume Delta is calculated between move K and move K+N with a search depth less than N, say N-D. The move for K resulted from the evaluation of board position for move K+N-D. What values of Delta are needed to indicate that the wrong move was made at K and not some prior move? Will the values of Delta indicate that? If it can be determined that the wrong move was made at K, then should the evaluation function be modified at move K, or at move K+N-D, or both, as the evaluation value for move K was backed-up from K+N-D

Can any correlation be determined between values of Delta as the search depth is increased; what patterns or trends exist, if any?

The values determined by selection are determined over a short time span, the past game. A long term average of the values should be tracked.

Since alpha-beta deals with a variable lower bound, the current author wonders if "deep cutoffs" only occur when the algorithm computes minimums. Cutoffs can occur on plys two and three, but those cutoffs occur on both branch-and-bound and alpha-beta. Or does the alternating between plus and minus between plys work with the line *if (ans >= beta) return(ans);* to address "deep cutoffs" when computing maximums?

In the current author's opinion, an 'archeological dig' should be performed on Samuel's work before it might be lost forever. The article does not reveal sufficient information to replicate his work.

BIBLIOGRAPHY

1. Botvinnik, M., Computers in Chess Solving Inexact Search Problems, Springer-Verlag, New York, 1984.

2. Frey, P. "Simulating Human Decision-Making on a Personal Computer," Byte, (July 1980) 56-72.(Vol. 5, No. 7)

3. Knuth, D. E. and Moore, R. W., "An Analysis of Alpha-Beta Pruning," Artificial Intelligence 6 (1975) 293-326.

4. Holland, J. "Genetic Algorithms," Scientific American, 267 (July 1992) 66-72.

5. Horowitz, E. and Sahni, S., Fundamentals of Data Structures, Computer Science Press, Inc. California, 1976.

6. Lee, K. and Mahajan S., "The Development of a World Class Othello Program," Artificial Intelligence, 43 (1990) 21-36.

7. Maggs, P. "Programming Strategies in the Game of Reversi," Byte, (November 1979) 66-79.(Vol. 4, No. 11)

8. Millen, J. "Programming the Game of Go," Byte, (April 1981) 102-120. (Vol. 6, No. 4)

9. Newborn, M., Computer Chess, Academic Press, New York, 1975.

10. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," IBM Journal, 3, 210 (July 1959).

11. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress," IBM Journal, 11, 601 (November 1967).

12. Seale, M. "Studies in Machine Learning using Game Playing," Master Thesis, Oklahoma State University, 1985.

13. Shannon, C. E., "Programming a Computer for playing Chess," Philosophical Magazine 41, 256 (March 1950).

14. Von Neumann, J., and Morgenstern, O., Theory of Games and Economic Behavior, Princeton University Press, Princeton, New Jersey, 1944.

VITA

ROD KEVIN MCABEE

Candidate for the Degree of

Master of Science

Thesis: PARTIAL ANALYSIS OF OTHELLO

Major Field: Computer Science

Biographical:

Personal Data: Born in Hobart, Oklahoma, On October 31, 1954, the son of Alvin and Helen McAbee.

Education: Graduated from Hobart High School, Hobart, Oklahoma in May 1972; received Bachelor of Science degree in Mathematics and a Bachelor of Science degree in Computer Science from Oklahoma State University, Stillwater, Oklahoma in May of 1978 and May of 1982, respectively. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July 1994.

Experience: Owned and operated small businesses; employed as factory worker and Quality Analyst/Engineer; employed by Oklahoma State University, College of Engineering, Architecture and Technology as a Manager for the College computer facilities.