1-1-1986

BILL : a table-based, knowledge-intensive othello program

Kai-Fu Lee
Carnegie Mellon University

Sanjoy Mahajan

CMU-CS-86-141

BILL: A Table-Based, Knowledge-Intensive Othello Program

Kai-Fu Lee and Sanjoy Mahajan

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

April 1986

Abstract

A constant dilemma facing game-playing programs is whether to emphasize searching or knowledge. This paper describes a world-championship level Othello program, BILL, that succeeds in both dimensions. The success of BILL is largely due to its understanding of many important Othello features by using a pre-compiled knowledge base of board patterns. Because of this pre-compiled nature of its knowledge, BILL's evaluation function simply consists of a series of table lookup's. It is therefore very fast. Additional key features of BILL include an iteratively-deepened zero-window search, an intelligent timing algorithm, an efficient linked-move killer table, and a hash table. This paper contains detailed descriptions of the game of Othello and BILL, the results of BILL, and an outline for future research.

This research was partly sponsored by a National Science Foundation Graduate Fellowship. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or the US Government.

Table of Contents

List of Figures

List of Tables

1. Introduction

Othello was invented in Japan in 1971 and is based on the game Reversi. Since its introduction to the United States, Othello has become a popular game for computer implementation. There are several reasons for this:

1. The rules are simple.
2. The board changes drastically throughout the game, making deep searches very difficult for human players.
3. The branching factor, or the number of legal moves per turn, is considerably lower than in chess, where most research effort in game-playing programs has been invested.

Therefore, unlike for chess or go, it is not difficult to build an Othello program that plays a reasonable game.

However, almost all Othello programs can be classified as one of two types: knowledge-intensive and slow or knowledge-deficient and fast. True understanding of Othello strategies requires the recognition and analysis of many board patterns. Knowledge-intensive programs attempt to recognize these patterns in their evaluation function, which is very time consuming. As a result, although these programs often play at the expert level, they were only able to search to a moderate depth. On the other hand, other programs use evaluation functions that are fast to compute could search several plies deeper. Nevertheless, the validity of the information in their evaluation functions is questionable. This lack of knowledge resulted in programs that could defeat casual Othello players, but were resoundingly defeated when pitted against more knowledgeable programs or players [1].

In this paper we present an Othello program, BILL, that uses a pre-compiled knowledge base capable of recognizing and evaluating complex patterns using only table look-up's. As a result, BILL's evaluation function contains more information than the knowledge-intensive programs. Yet its speed is comparable to the knowledge-deficient programs because its evaluation consists only of a series of table look-up's. Its performance is further enhanced by a number of state-of-the-art artificial intelligence techniques, including an iteratively-deepened, zero-window alpha-beta procedure, a hash table, a linked-move killer table, a two-phase end-game search, and an efficient usage of opponent's time.

This combination resulted in a formidable Othello player. BILL captured first place in the Waterloo Computer Othello Tournament, and second place in the 1986 North American Computer Othello Championship. This second place finish was partly a consequence of an unfortunate draw of colors. As further evidence of its ability, BILL consistently defeats IAGO, the program that inspired BILL. Since IAGO was generally considered as good as, if not better than, the best human players [2], BILL is clearly one of the best Othello players in the world.

Chapter 2 contains a description of the game of Othello and a discussion of strategy. The Othello program BILL is described in detail in chapter 3. Chapter 4 describes BILL's history and tournament record. Chapter 5 outlines BILL's weaknesses and areas of current and future research. Finally, Chapter 6 contains some concluding remarks.

2. The Game of Othello

2.1 The Rules of Othello

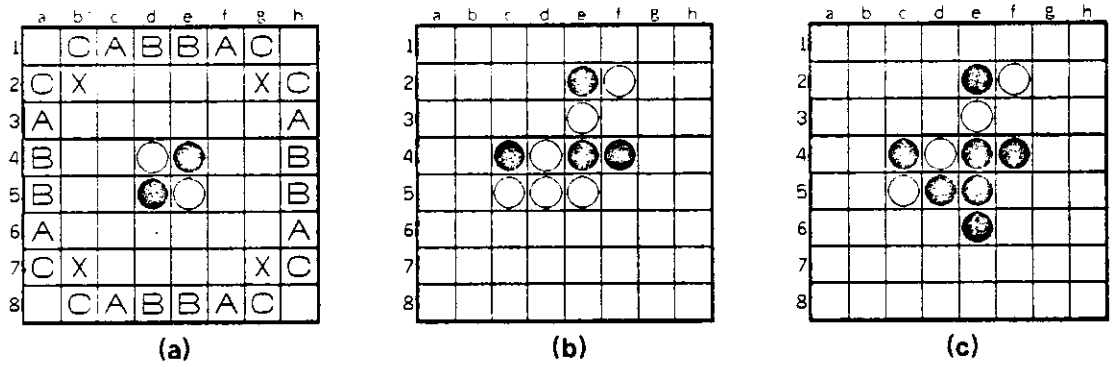


Figure 2-1: (a) shows the initial Othello board set-up and the standard names of the squares; (b) shows a sample board with legal moves (for Black) to C6, D6, D2, E6, and G2; (c) shows the board after Black plays to E6.

The rules of Othello are quite simple. The game is played on an 8 by 8 board, initially set up as in Figure 2-1(a). Each player, starting with Black, takes a turn by placing a piece of his color on the board, flipping to his own color any of the opponent's pieces that are *bracketed* along a line. There are two restrictions, however: (1) one of the bracketing pieces must be the piece just placed on the board and (2) a move must flip at least one piece. Figure 2-1(b) contains a sample board showing with the legal moves, and Figure 2-1(c) shows the board after the move. When a player does not have any legal moves, he must pass his turn; when neither player has a move, the game is over and the player with the most discs is declared the winner. The game usually ends when all sixty-four squares are occupied, but this is not a requirement.

Standard Othello notation consists of naming a square by a letter-number combination. The letter (A-H) indicates the column, and the number (1-8) indicates the row. For example, the lower left corner is named A8. Some of the more important squares types are also given designations. For example, the square on the edge that is next to a corner is a C-square, while the square diagonally adjacent from a corner is an X-square. Figure 2-1(a) shows the standard names of the Othello squares.

2.2 Othello Strategies

The most important Othello strategy, upon which all others are ultimately based, is that of disc stability. A stable disc is defined as one that cannot ever be flipped. Acquisition of stable discs leads to victory for two reasons:

1. Stable discs can be used to create more stable discs.
2. Stable discs are the deciding factor at the end of the game, when all discs are, by definition, stable.

Thus, any component of a strategy must consider long-term disc stability. In this section, we will first discuss an incorrect strategy used by novices. Next, we will discuss several effective strategies used by experts to maintain and increase their stable disc count.

2.2.1 The Maximum Disc Strategy

The correct Othello strategies are by no means intuitively obvious. Many beginners make the plausible but incorrect assumption that, since maximum disc count is the *final* goal, the move that flips the largest number of discs is the best one. This strategy, known as the *maximum-disc* strategy, results from not understanding the value stable discs. What the maximum discs strategy accomplishes is the quick acquisition of many unstable discs, at the cost of giving the opponent a large number of stable discs at the end of the game. Thus, this strategy leads to ruin.

Having a large number of unstable discs is harmful because it provides the opponent with a large number of moves. Figure 2-2 shows an extreme position resulting from playing the maximum disc strategy. In this example, White greedily took 35 discs, leaving Black with just one. However, this gave Black a large number of moves, leaving White with only a few. A few moves later, White will be forced to surrender an edge because of his lack of mobility. The importance of mobility will be discussed at length in Section 2.2.4. So poor is the maximum-disc strategy that many Othello experts advocate playing a *minimum-disc* strategy throughout the late opening and midgame.

Figure 2-3 shows the disc differential in BILL's games against IAGO and at Waterloo. In the late opening and mid-game, BILL consistently has about 5 discs more than IAGO, and won both games. On the other hand, BILL often had 10 discs fewer at Waterloo, and won all four games with overwhelming margins. From this we can conclude that while the minimum-disc strategy is not almost useful, the maximum-disc strategy played by other programs at Waterloo is undoubtedly unsound.

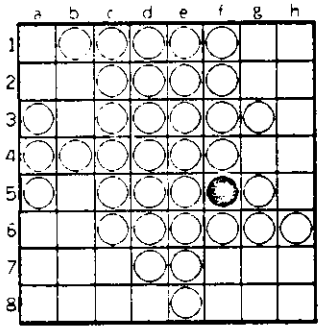


Figure 2-2: Example of how the maximum disc strategy leads to ruin: in (a), from BILL (B-53) vs. IAGO[Burton] (W-11), BILL only has 1 disc, while IAGO has 35. Black to move.

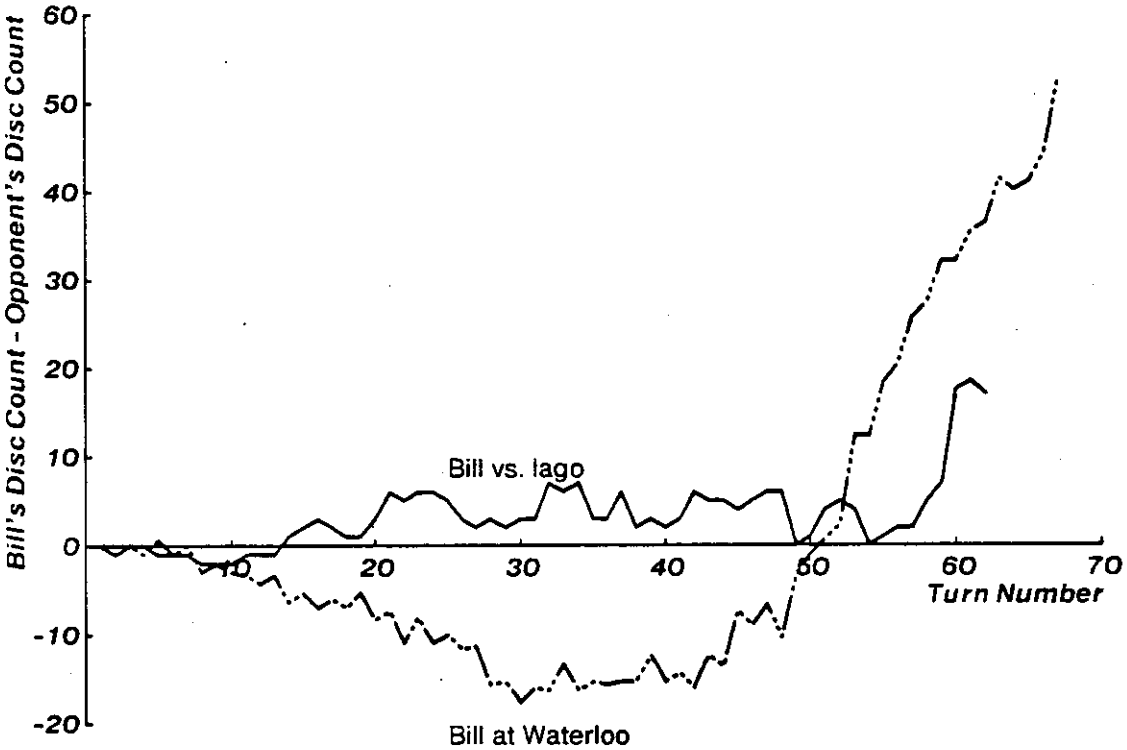


Figure 2-3: Disc count differential throughout the games (All games were won by BILL)- This illustrates the lack of information in the maximum-disc strategy, as played by the other programs at Waterloo, all of which lost to BILL by overwhelming margins. It also illustrates the nebulous value of a minimum-disc strategy since IAGO, which maintained a lower disc-count, also lost both games. There are moves beyond turn 60 because the opponent was forced to pass, thereby creating more moves for BILL.

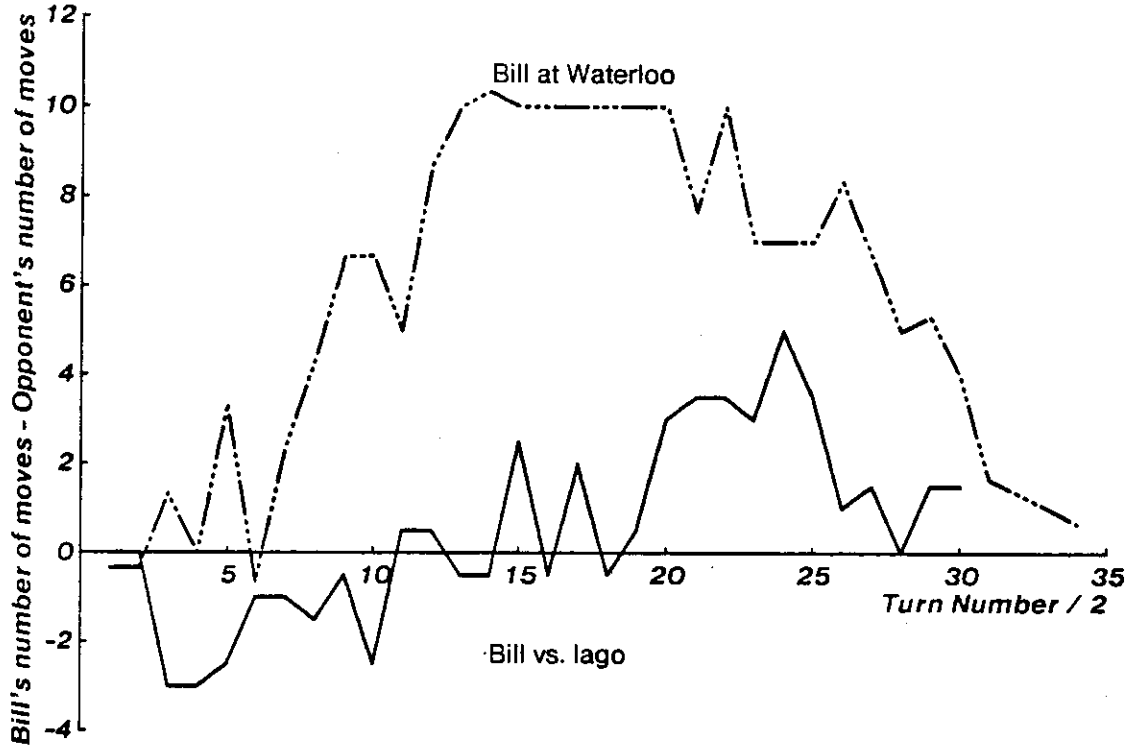


Figure 2-4: Move differential throughout the games (All games were won by BILL). This illustrates the importance of having enough alternatives so as not to be forced into making a poor move.

2.2.2 Edge Control

After playing the maximum disc strategy for a while, most beginning players discover the value of corner and edge discs. These discs are important precisely because of their stability. Edge discs can only be flipped in one way (along the edge), while corner discs cannot be flipped at all. In addition, if an edge disc is adjacent to a stable edge or corner disc of the same color, it becomes stable. Thus, stable edge discs and corner discs be used as anchors to create even more stable discs, leading to eventual victory. Most edge play, and therefore Othello play in general, revolves about the gain and loss of corners.

Relying on these observations, most beginners then play the "edge greedy" strategy of playing to all available edge squares and not moving out of the central 16 squares (the "sweet-16"), lest they present the opponent with an opportunity to gain access to an edge. Edge play is much more complicated than that simple strategy would indicate, however. Numerous pitfalls await the edge greedy player. He can fall into a number of edge traps, or can fall prey to reduced mobility.

2.2.3 Edge Traps

Edge traps are one way to gain or lose a corner. One of the most dangerous such traps that results from occupying a lone c-square[1] is shown in Figure 2-5(b), where Black can win the corner if he has sufficient access to the northern edge. To see this, consider the position with Black to move after White just moved to g1 and Black responded with c1. Now White has four choices: move to d1, e1, f1, or not move on the edge at all. If White moves to d1 or to e1, Black responds with f1, winning the corner. If White moves to f1, Black responds with e1, again winning the corner. If White does nothing at all, Black will win the corner by moving to e1.

Another edge trap is the *unbalanced edge.* An unbalanced edge (shown in Figure 2-5(c)) is one that contains five discs of the same color, but not other discs. This position is inferior because it gives White a free move to the x-square at b2. If Black responds by taking the corner, gaining one stable disc, White can most likely move to b1 and later h1, gaining 7 stable discs. Even if Black moves elsewhere, he has created a square which is safe only for White. Since many similar edge traps exist, a proper understanding of edge positions is necessary to assuring stability.

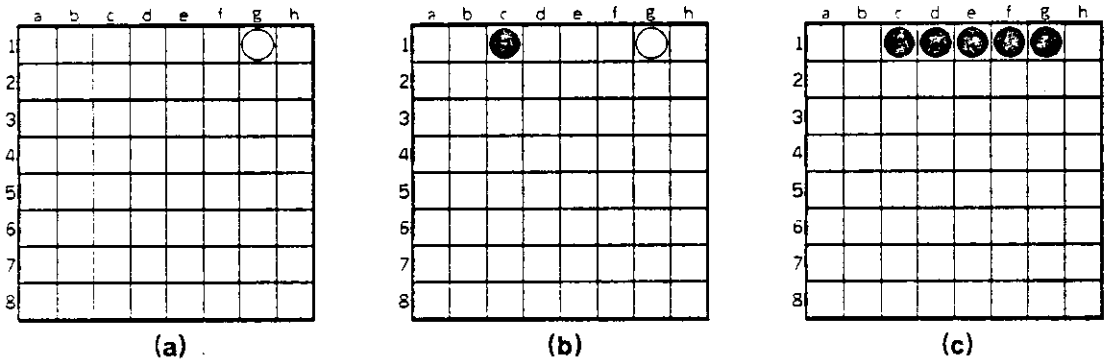


Figure 2-5: Example of edge traps: (a) shows a poor edge move (g1) by White because Black can move to c1, as shown in (b), which guarantees Black's possession of the h1 corner if Black has access to the necessary edge squares; (c) shows an *unbalanced edge*, another dangerous edge position.

[1] For a diagram showing the square naming conventions, see Figure 2-1(a).

2.2.4 Mobility

The most difficult strategy for beginners to discover is that of mobility optimization. Mobility is related to the number of moves a player has, and is a critical Othello strategy. Since only a very foolish opponent would willingly allow one to create stable edge discs early in the game, he must be forced to yield these stable edge discs. A means to this end is mobility optimization. Figure 2-4 shows the importance of having enough moves and the tragic consequences of not having enough. In the Waterloo Othello tournament, many of the programs there did not properly evaluate mobility and quickly fell behind. IAGO, on the other hand, fully appreciates the importance of mobility.

One method of measuring mobility is simply counting the number of moves one has. IAGO uses a method similar to this. However, it is also important to consider the worth of each move. The proper judgment of the goodness of each move is what distinguishes the expert from the amateur. This judgment is essential, as it has been noted that in expert play that the first player to run out of good moves usually loses. To decide on the worth of a move, a number of factors must be taken into consideration. Immediate harmful consequences provide the most important consideration. However, long-term results of a move must also be considered. As we will see later, moves that flip many discs, or flip discs in many directions are inferior because they reduce a player's future mobility. Moves that lose a corner are almost worthless and should not be given full weighting in a move count. Figure 2-6 shows two examples of how inattention to this factor can be disastrous. In both positions, IAGO had many moves but overestimated its positions because of its simple move-counting evaluation. This evaluation did not consider the fact that most of IAGO's moves give up a corner. Thus, IAGO was actually behind in mobility and should have lost both games[2].

The number and type of discs flipped by a move also affect its worth. In general, the fewer discs a move flips, the better; in addition, the fewer directions that discs are turned, the better. These facts follow from the principle of minimizing the opponent's mobility. Turning too many discs provides possible legal moves for the opponent. Based on an analysis of 80 tournament games, Anders Kierulf, a world-class Othello player, estimated that in games between expert players the average number directions flipped was 1.1 at move 10 and only 1.5 at move 50 [3].

Finally, one should avoid flipping discs that are next to empty squares, or *frontier* discs. Flipping many frontier discs results in a *wall*, a sequence of frontier discs of only one color. Walls limit one's mobility while increasing the opponent's mobility. Figure 2-7 contains examples from BILL's games showing how walls adversely affect mobility. Arnold Kling, an Othello expert, considers the concept of wall-avoidance so important that he has defined a special type of move called the Perfectly Quiet Move, a move that flips only one internal disc [4]. It can be seen now that Othello strategy is more complicated than it first appears.

---

[2] In reality, IAGO won the game against ALDARON by a fluke [2]

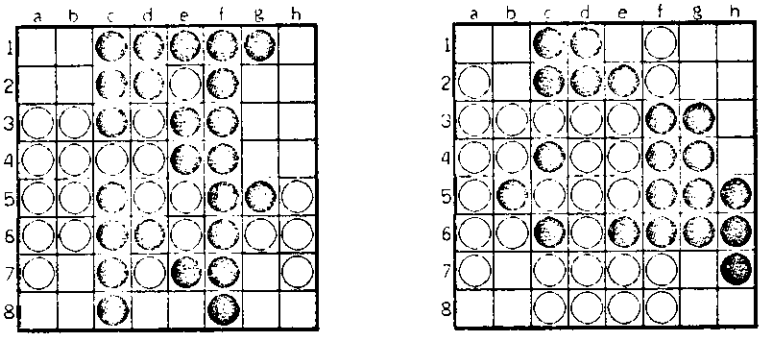


Figure 2-6: Examples from IAGO's games showing how important *weighted* mobility, as opposed to just move counting, is: in (a), from ALDARON (B-30) vs. IAGO (W-34),[3] 7 of IAGO's 11 moves give up a corner; in (b), from IAGO (B-23) vs. BILL (W-41), IAGO has 7 moves, of which 5 give up a corner immediately.

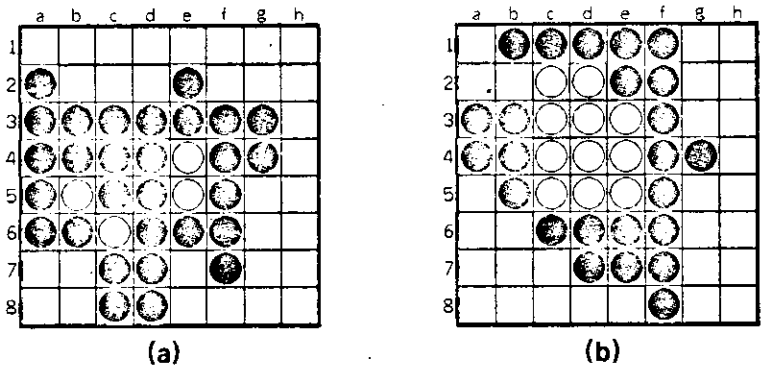


Figure 2-7: Examples of the lack of mobility created by walls: in (a), from OTHELLO[0] (B-10) vs. BILL (W-54). Black has one legal move, while White (to move) has 12; in (b), from GRAY BLITZ (B-4) vs. BILL (W-60), Black has 2 moves, while White (to move) has 15.

[3] This is the game IAGO won by a fluke [2].

3. Bill

3.1 Search, Timing, and Ordering

3.1.1 Forward Pruning

Forward pruning prunes nodes *without* having searched them. This is believed by many to be very dangerous [5] because the reward of a good move is often realized many plies later. Without searching, one cannot be certain that a move is bad.

However, we have several reasons to believe that limited forward pruning is appropriate in the game of Othello:

1. Moving to an X or a C-square at an early stage is generally considered unsound by Othello experts [4].
2. There are a number of edge configurations that make an X or a C-square move acceptable. Without these configurations, an X-square move is unlikely to be sound.
3. Occasionally in the early parts of the game, BILL would prefer an X or a C-square move if it is far better than the other moves in terms of mobility.[4] The danger of an X-square move may not be obvious to BILL until many moves later.

Whenever an X-square is encountered in the search tree, BILL decides whether it should be pruned based on the following rules:

1. If the total disc count is 35 or more, never prune.
2. Otherwise, if the adjacent corner is occupied, do not prune (because the occupation of the corner nullifies the problem of an X-square move).
3. Otherwise, if both adjacent C-squares are occupied, always prune (because that makes it impossible to create any stable discs for the player to move, and hence is never desirable).
4. Otherwise, prune if and only if the total disc count is less than 25.

C-square safety is more expensive to determine, and is more likely to be harmful. Consequently, only top-level C-square moves are pruned when:

1. The adjacent corner and A-square are both empty.
2. There is no opponent's disc on the nearest B-square.
3. The total number of discs is less than 28.

---

[4] As will be shown in Section 3.2.1, an X-square move is penalized significantly in the edge table. So, it will be preferred only when the mobility gained is overwhelming.

Although many nodes are pruned in the early parts of the game, the pruning of these X and C-square nodes is unlikely to reduce the time requirements significantly, because the X and C-square moves are likely to be pruned by the alpha-beta search anyway. Our main goal in forward pruning the X and C square moves is to prevent BILL from making them in the earlier parts of the game. This goal is achieved by the conservative forward pruning of early X and C square moves.

3.1.2 Iterative Deepening

BILL uses an iteratively-deepened, alpha-beta search [5]. An iterative deepening search always performs an N-ply alpha-beta search before attempting an N+1-ply search. At the beginning of each turn, BILL performs an alpha-beta search to level MAX (2, *LastLevel* - 2), where *LastLevel* is the level searched to in the last move. Then, if the timing algorithm permits, it increments its search level by one.

Iterative deepening has two important advantages:

1. It facilitates timing control. When BILL has used too much time for the current move, it could always abort the search and use the best move from the previous level. In a fixed-depth alpha-beta search, the program must either risk defaulting on time or return a move based on partial information.
2. Important ordering information is stored in the hash table and killer table. This information will reduce the search in later iterations as well as later moves.

3.1.3 Timing

Some programs simply use a fixed-depth search [6] [3]. However, due to the strong dependence of the alpha-beta search on ordering, a well ordered n-level search may require only a very small percentage of the time needed by a poorly ordered n-level search. Thus, using a fixed-level search leads to inefficient use of time, and creates the possibility of defaulting on time. For the same reason, it is not sufficient to simply check the clock after every level of iterative deepening. Even checking the clock after each top-level branch of the search is not sufficient. It is possible for a single branch to take much longer than expected, as the number of nodes in an alpha-beta tree has a very high variance. If this happens, the possibility of defaulting on time materializes. Since BILL was designed to play in a competitive environment, it must use a more sophisticated algorithm. The algorithm described below is heavily influenced by the timing algorithm of Hitech [7], the winner of the 1985 ACM Computer Chess Championship.

International Othello rules allow 30 minutes for a player to make all of his moves. To divide the time among each move, a time allocation fraction is assigned to each move, based upon its relative importance. BILL's allocation fraction is shown in Figure 3-1. As the game progresses, more time is allocated. Since BILL's opening book always covers the first 2 moves for either side, these moves are assigned time fraction of 0. If

BILL used more than 2 moves from the opening book, the saved fraction is redistributed among the remaining moves. Since BILL is always able solve the endgame 12 moves from the end, the last 11 moves are given little time. Although there are 60 moves for both players in a game, one may have to make more than 30 moves if the opponent has no moves. Thus, BILL allocates time for 40 moves, although the 10 additional moves are assigned very little time.

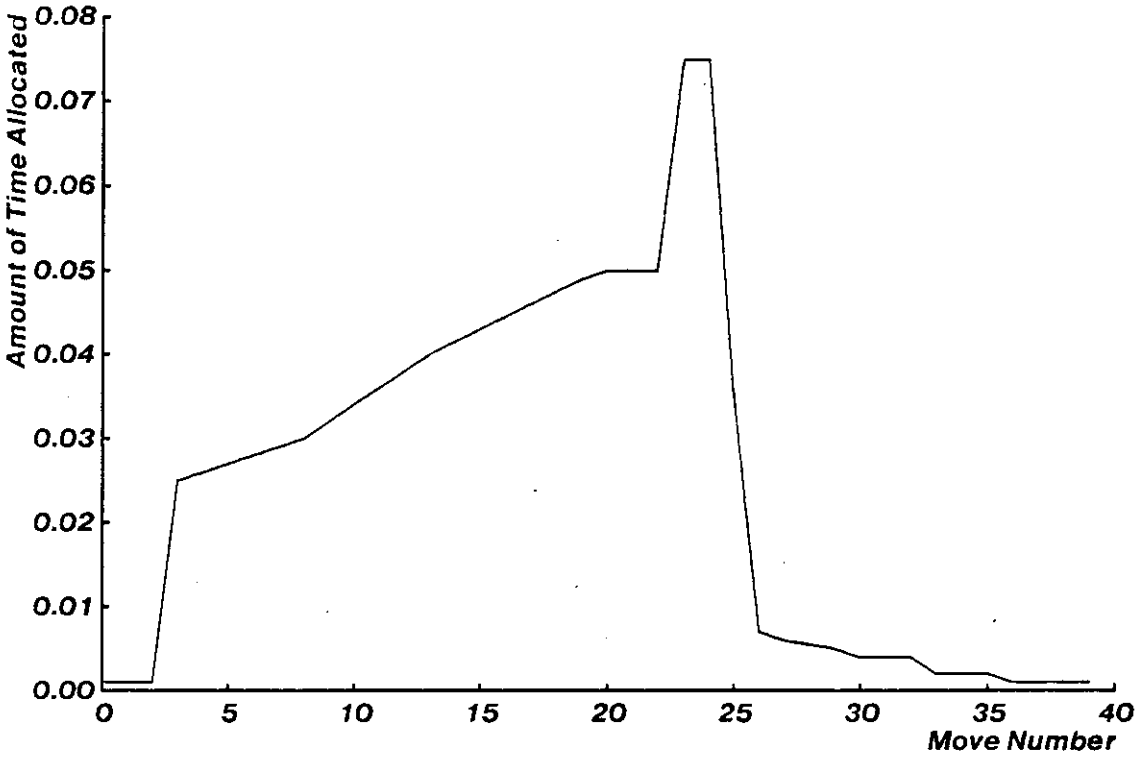


Figure 3-1: Time allocation fraction for BILL.

Before each move, BILL computes the allocation for this move by multiplying the time left on BILL's clock by the allocation fraction for this move and divided by the total remaining fraction. BILL uses this time allocation to determine when to terminate searching and make the best known move. Whenever BILL finishes one iteration of search, it checks the following conditions:

- If less than 40% of the time allocation is used, begin the next iteration.
- If the time elapsed is between 40% and 100% of the time allocation, continue only if the last two levels of search disagree as to the best move.
- If more than the time allocation is used up, stop.

Since the branching factor of BILL is about 3.7, whenever the time elapsed falls between 40% and 100%, BILL is expected to use more time than allocated. This only occurs when the last two levels disagree, which does not happen very often. Consequently, the extra time used here is balanced by the time saved when BILL decides to stop early.

As a final precaution, if BILL has exceeded its allocated time by more than 80%[5], an internal alarm goes off. This occurs several times in a typical game when the ordering is poor[6] and when the last two levels disagree. In the case of search termination due to alarm, a partial level (such as 7.4) is recorded for BILL.

In an end game search, there is no iterative deepening to check the time used. Furthermore, 180% of the time allocation may not be sufficient to complete the search. Therefore, whenever BILL decides to search to solve the endgame, BILL sets the internal alarm by leaving just enough time to perform endgame searches for its subsequent moves. See Section 3.1.4.3 for more discussion.

In BILL's tournament games, it is given 25 minutes to make all its moves (5 minutes is needed to transmit and enter the moves). In the 8 official and 1 unofficial games of the North American Computer Othello Championship, BILL had an average of 180 seconds left at the end of each game. Although we were permitted to modify BILL's clock at BILL's request, we found that BILL's management of its own time so satisfactory that it was not necessary to change it at all during the 9 games at the North American Computer Othello Championship. Therefore, this timing algorithm has proven effective in utilizing the total time while ensuring that BILL does not default on time.

3.1.4 Search

Search is at the heart of modern computer game-playing programs because it enables programs to see more than their evaluation function knows. For example, search reduces the pathological problem of edge interaction to only a nuisance. Computers excel at searching, and BILL is no exception. BILL uses a modified, iteratively deepened alpha-beta search known as zero-window search.

3.1.4.1 Zero-Window Alpha-Beta

The speed of an alpha-beta search depends heavily on the number of alpha and beta cutoffs found. Thus, any method that increases this number, even if it sacrifices some knowledge about the exact value of a position, is potentially desirable. Zero-window search, similar to Scout search [8], is such a method and is used by both the normal search and the endgame search. First, the normal zero-window search and then the endgame zero-window search will be described.

---

[5] Unless there was "very little time" left to begin with. In that case, the alarm is set at 100%, rather than 180%, of the time allocation.

[6] One cannot always achieve good ordering, for in that case search would be unnecessary.

3.1.4.2 Normal Search

In each iteration of the zero-window search, the top-level moves are first ordered as described in Section 3.1.5. The first choice move is searched with a *narrow window* based upon the results of the last iteration of searching. This search establishes an exact value, which is used as a *zero-window* for searching the remaining moves. The advantage of using a narrow- or zero-window, instead of the usual ∞ to $-\infty$, is that it increases the number of alpha-beta cutoffs, thereby increasing the effective search depth.

Using such a window is a calculated risk, however. The risk is that the value returned by a search may fall outside the window. In this case the search yields incomplete information. It is simple to determine if that is the case. If the value returned by the search is equal to the upper bound of the window, then the true value is at least that. This situation is known as *failing high.* Conversely, if the value returned is equal to the lower bound, the true value is at most that. This situation, naturally, is known as *failing low.* Upon either failing high or low, there are two options: use the knowledge gained already and stop searching the current branch, or adjust the window size and re-search, hoping the true value is inside the new window. If the actual value is outside the narrow window for the the first top-level branch, BILL always chooses to re-search. This is because the rest of the zero-window search depends on knowledge of the exact value of the first, and hopefully the best, branch.

Once a value is found for the first node, the other branches are searched with a *zero-window.* The lower bound of this window is the exact value found for the first branch, while the upper bound is only one more than the exact value. Thus, this search is only capable of determining whether the move under examination is better or worse than the current best move. If a move is worse than the best move, it is ignored and the next move is searched. If the move is better, however, it is made the new best move. Even though no exact value for it is known, there is no problem unless another move that is better than the upper window bound is found.

If two such moves are found, both moves are searched with an upper bound of ∞ to get exact values. The better of the two is then made the current best move. Searching the top-level moves continues this way until either all the moves have been searched and a best move is known, or until search is aborted by the alarm (See Section 3.1.3). If there is still time left, another iteration of iterative deepening begins.

3.1.4.3 Endgame Search

The endgame search, which also uses narrow- and zero-windows, is a very important search because it is capable of returning the exact value of the game by searching to the end of the game. There are two possible evaluation functions that can be used when the game has ended:

1. A simple win/loss/draw function. The search which uses this function is called an *outcome* search.

2. A disc count differential function. The search which uses this function is called an *exact* search.

The *outcome* search is very fast because it leads to significantly more alpha-beta cutoffs. However, it is less accurate than the *exact* search in that it does not provide the margin of victory or defeat.

Because the first function has only three possible values, a window with a lower bound of loss (-1), and an upper bound of win (+1). If BILL has not yet searched to the end, BILL will do so when the timing algorithm determines there is enough time to complete an *outcome* search to the end. The timing algorithm makes an estimate of the time required to complete the *outcome* search based on the number of empty squares on the board. Usually a *outcome* search is completed with 15 or 16 empty squares. The alarm for an endgame search is set so that there will be at least enough time to complete an endgame search on the player's next move.

The *outcome* search is conducted like a normal zero-window search. A value for the first move is obtained from a one-window of win/loss/draw. The result of that is used to make a zero-window for the other branches. If a winning line is found, BILL terminates the *outcome* search and makes a decision to either stop searching and make the move, or to do a *exact* search.

Once BILL has searched to the end, subsequent searches will always be endgame searches. If BILL has already completed an *outcome* search and there is enough time for an *exact* search, BILL will search to the end using the *exact* evaluation. The *exact* search attempts to maximize the difference between BILL's and the opponent's disc count. The first move is searched with a narrow window, which is determined by the evaluation of the last normal search and the result of the *outcome* search. Subsequently, all branches are searched with a zero window. BILL can usually complete an *exact* search with 13 to 15 empty squares on the board. This idea of a two-phase endgame search is taken from IAGO [2], although BILL's control strategies are more complex.

3.1.5 Search Ordering

In order to realize the full power of the alpha-beta search, it is important to order the legal moves so that they are examined as close to the optimal ordering as possible. Ordering is even more crucial in the zero-window search because of the extra time needed for failing high. BILL maintains two data structures, the *hash table* and the *killer table*, to improve its ordering of the nodes.

BILL's moves are generated *incrementally* as needed. When BILL needs to sprout a node in the alpha-beta search, the empty squares are scanned in an order indicated by the *hash table* and the *killer table* for legal moves. As soon as a move is found, the scanning is halted, and that move is returned. This has two major advantages:

1. Move generation is a time-consuming process [2]. By generating the moves as needed, every alpha-beta cut-off saves time for evaluation as well as time for legal move generation.
2. If all moves were generated for each node, it may be necessary to sort them, which is another expensive operation. Using incremental move generation, sorting is not necessary.

In the next two sections, we will discuss how the two tables are organized and used.

3.1.5.1 The Hash Table

Given an encoding of the current state (the board position, the color to move, and the number of discs on the board), the hash table tries to suggest a best move to expand first. Whenever the hash look-up is successful and the move returned is legal, that move is expanded first.

The best move was stored in the hash table when the same state was encountered, and the best move was found by searching, in the alpha-beta procedure previously. This could be either from a previous iteration in the iterative deepening or from searching during a previous move. When a new best move is put into the hash table, all outdated entries are deleted; therefore, the move found was inserted by the deepest search from that state.

The hash table contains a 15-bit key, which is used to index into the 2^{15}, or 32768, hash table entries. A 16-bit hash lock, the color to move, and the number of discs on the board are also used to differentiate states which share the same key. Because of the large number of possible Othello configurations, it is possible that two different states will have the identical key, lock, color, and discs. Therefore, after the best move is retrieved from the hash table, it must be tested for legality.

The hash table contains exact information from a previous search about this position; therefore, it is our most reliable source of ordering information.

3.1.5.2 The Killer Table

If one is always able to examine the best descendant of a node first, the alpha-beta (and zero-window) search is guaranteed to be optimal [9]. The hash table attempts to provide information as to the best descendant; however, there are three ways it can fail: (1) the node had not been seen before, so the hash table contains no information, (2) the best move provided by the hash table is non-optimal after additional search, and (3) the best move provided by the hash table is illegal. In any of the three case, it is still very important to examine the descendants (or remaining descendants) in an order as close as to optimal as possible. BILL's killer table attempts to order all the moves heuristically.

The killer table contains 60 entries for each color. The entry x in the table for color c represents

responses for c to the move x by the opponent. This entry contains not one move, but a linked list of *all* the empty squares. They are heuristically ordered from the best response to the worst response by c to the move x.

This structure is complementary to the incremental move generation process. Since finding the legal moves requires scanning of all empty squares that are next to a disc, the linked list provides an order in which to search for legal moves. Whenever there is a cut-off, the move generation and the linked list traversal are aborted.

We will now discuss how each linked list entry in the hash table is maintained. When the game begins, each empty square is on the linked list of every other empty square. This is initialized in a heuristic fashion. For example, for a move by Black to the upper-left x-square, best killers are probably: (1) the upper-left corner, (2) the two c-squares adjacent to the x-square, (3) the other corners, . . ., and lastly, the other x-squares.

Since the best responses to a move may vary from game to game, this static ordering will often be wrong. In order to maximize the pertinence of the killer table to a given game, we introduce the following algorithm for dynamic re-ordering of the nodes in the linked list. When a best move was found in response to another move, or when a move causes a cut-off to another move in the search, the linked list entry for that color and previous move is updated. The good response move to the previous move is moved up one in the linked list by swapping it with its predecessor in the linked list. This will almost always result in the propagation of good moves to the front of the list while leaving the bad ones in the rear.

The killer table data structure was designed with efficiency in mind. The linked list is implemented as a doubly linked list, which makes node-swapping a constant-time operation. In order to save traversal time, whenever an actual move is made, that move is deleted from every linked list in the killer table. But deletion in such a structure still requires O(n) time. Therefore, an array of pointers is maintained for each entry in the killer table. The nth entry in the array points to the location of the nth move in the linked list. This reduces the complexity for deletion to constant time as well.

This algorithm is somewhat different from most killer table algorithms [5]. In most killer table algorithms, a small value is added to a good response to a move. If we used such a structure, we would have two choices: (1) Sort all responses at every node, which requires O(n log n) time, where n is the number of moves, or (2) maintain the best few (not necessarily legal) responses, which provides only partial information. Instead, with the above algorithm, we exploit the fact that the possible moves always correspond to the empty squares remaining. Thus, a linked list structure allows us to order *all* empty squares heuristically. Since the list of empty squares must be examined for legal moves anyway, there is *no* additional cost for accessing the killer table.

3.1.6 Search Statistics and Experiments

3.1.6.1 Search Statistics

One method of evaluating the above methods is to examine the statistics maintained by BILL during its games. On a VAX-11/785, it can search to an average depth of 8.0 levels under international tournament time controls. This figure is a weighted average between an average depth of 7.7 levels for normal search and 15 levels for the first endgame search. BILL searches 1100 nodes per second and achieves a very favorable branching factor. BILL is capable of reducing a true branching factor of 10 to an effective branching factor of 3.7, which is very near the optimal value of 3.45. A comparison between these statistics and those of IAGO can be found in Appendix I.

3.1.6.2 Zero-Window Search vs. Normal Alpha-Beta Search

In order to determine the effectiveness of the zero-window search in reducing the branching factor, a controlled experiment was performed. Two versions of Bill were generated for this experiment. One version used BILL's zero-window search, while the other used a simple alpha-beta search. Each version was tested on the same 22 positions (taken from the game BILL lost to ALDARON, which is shown in Appendix II as Figure II-7). In order to make the branching factors comparable, timing was disabled, and each version searched each position to a depth of 8 before terminating the search. The results of the experiment are shown in Table 3-1. It can be seen that using the zero-window search leads to an improvement in the number of nodes searched and the branching factor. This improvement is particularly striking for greater depths.

	Zero-Window		Non-Zero-Window	
Depth	Branching Factor	Nodes	Branching Factor	Nodes
5	4.55 (0.37)	2075	4.69 (0.39)	2415
6	4.24 (0.40)	6490	4.32 (0.39)	7243
7	4.06 (0.39)	21380	4.28 (0.42)	31246
8	3.89 (0.36)	63809	4.11 (0.40)	102244

Table 3-1: The utility of the zero-widow search is illustrated by this table. The first number under each type of search is the effective branching factor, with the standard deviation in parentheses. The number of leaf nodes examined in searching to the specified depth is recorded next. There are 22 data points for each entry.

3.1.6.3 Killer Table Statistics

To demonstrate the effectiveness of our novel killer-table approach, we examined the location of the BILL's final choice of move in the killer table in all 15 games shown in Appendix II. Figure 3-2 shows how often BILL's final choice is ranked first, second, etc. in the killer table. It can be seen that the final choice is extremely likely to be near the beginning of the killer table. Since only legal moves are relevant for this experiment, the illegal moves in the killer list are not examined.

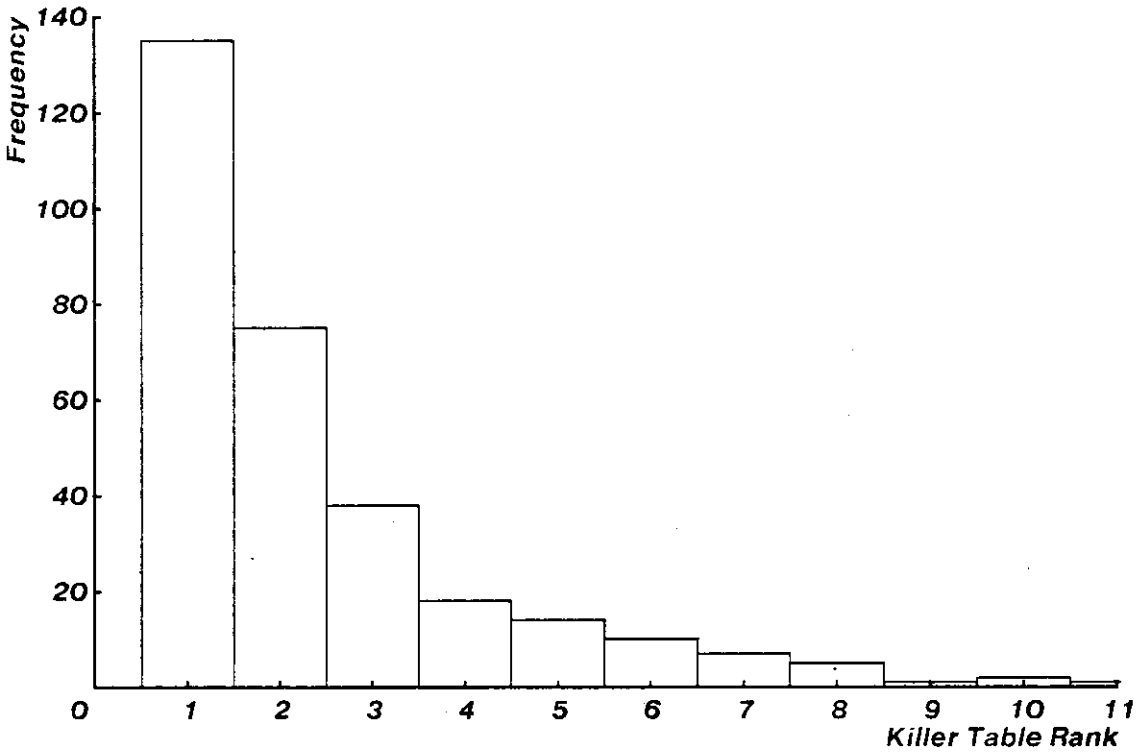


Figure 3-2: Ranks of BILL's final move in the killer table.

Figure 3-3 shows the average rank of the final move in the killer table as a function of turn number. Moves before 7 are usually made from the opening, and moves after 25 are always results of the endgame search. The killer table is not updated or needed in either case; therefore, only moves between 7 and 25 are used to evaluate the killer table. Figure 3-3 is interesting as it illustrates that the utility of the killer table increases after a period of uncertainty in the beginning.

3.1.6.4 The Effect of Killer and Hash Table on Branching Factor

The results in the previous section provided some interesting statistics; however, they do not provide a clear answer to the question "just how much is gained by having a killer table?" In order to answer this question, as well as one pertaining to the hash table, four versions of BILL each played a game against itself. The four versions corresponded to:

Figure 3-3: Average rank of BILL's final move in the killer table as a function of turn number in the game.

1. Use both the hash and the killer tables.
2. Use the hash table only.
3. Use the killer table only.
4. Use no hash table and no killer table.[7]

In this experiment, it was necessary that BILL played identical games so that the results are comparable. Therefore, timing control was disabled, and the results are obtained from fixed-depth searches.

The results are shown in Table 3-2. Branching factor is reduced significantly by the use of the killer table ($p < .0001$ for a 6-ply search, and $p < .0002$ for an 8-ply search) or hash table ($p < 10^{-9}$ for a 6-ply search, and $p < 10^{-7}$ for an 8-ply search), with hash table being more valuable. The use of *both* the hash and killer tables resulted in additional improvement ($p < .002$ and $p < .003$ for 6 and 8-ply searches when compared against killer table only, and $p < .04$ and $p < .09$ against hash table only). In the case of an 8-ply search, the version without either table takes 7 times as much time as the one with both tables. This justifies our use of a novel killer table structure.

[7] Although this version has no dynamic ordering, a static ordering is still obtained from the original killer response list (examining corners first, x-squares last, etc).

Table Used		Effective Branching Factor	
Hash	Killer	6-ply search	8-ply search
Yes	Yes	3.91 (0.35)	3.60 (0.32)
Yes	No	4.09 (0.37)	3.74 (0.37)
No	Yes	4.20 (0.44)	3.85 (0.41)
No	No	4.69 (0.46)	4.26 (0.52)

Table 3-2: The effect of using hash and killer tables in Bill. The numbers shown are the effective branching factors with the standard deviation in parenthesis. Each entry is based on 36 data points.

3.2 The Evaluation Function

Because of the difficulty of creating and tuning non-linear evaluation functions, BILL's evaluation function for non-terminating board positions is a linear combination of three terms:

$$EVAL(board,player) = EC \times Edge\ Advantage + MC \times Mobility\ Advantage + SC \times Occupied\ Square\ Advantage$$

$EC = 500$
$MC = 350 - 2 \times Discs$
IF $Discs < 10$
    $SC = 200 - Discs$
ELSE IF $Discs < 20$
    $SC = 190 - 2 \times (Discs - 10)$
ELSE IF $Discs < 40$
    $SC = 170 - 5 \times (Discs - 20)$
ELSE IF $Discs < 50$
    $SC = 70 - 7 \times (Discs - 40)$
ELSE
    $SC = 0$

where *Discs* is the number of pieces on the board. *EC*, *MC* and *SC* are slow-varying application coefficients [10]. These coefficients were determined by a round-robin tournament among 10 different sets of coefficients of BILL.

The edge advantage term is normalized to the range [-1000, 1000]. Theoretically, the mobility and Occupied-Squares terms could be much larger than 1000 or smaller than -1000; however, in actual games, they always lie in [-1000, 1000].

Edge advantage measures the edge position of the player. Mobility advantage measures *weighted*

current mobility and *weighted potential mobility* for the player. Occupied-Squares advantage measures *sequence* and *weighted square* for the player. All three are computed using table look-up's. In the next two sections, we will discuss how these tables are generated and used.

3.2.1 Edge Table

In Section 2.2, we saw the importance of understanding edge positions. We cannot rely upon search to uncover the dangers of certain edge positions. An example with C-square moves was shown in Figure 2-5. Moving to the C-square on some edge configurations is inadvisable; however, it may require as many as 30 plies to see that the opponent could take the corner. Moreover, the true danger of such a move depends on the probability that the opponent could make certain moves.

Fortunately, there are only 8 squares on each edge; thus, we could assign a value to each of the 3^8 (each square could be blank, White, or Black) , or 6561, edge patterns. However, our experiments showed this to be inadequate because the two X-squares adjacent to the edge play an important role in edge evaluation. For example, consider a position from the game BILL (B-61) vs CASSIO (W-3), shown in Figure 3-4(a). The northern edge is not bad for White. In fact, he has a tremendous overall advantage due to mobility. As long as White could not play into a northern corner, this edge serves as an anchor for Black. However, consider the position in Figure 3-4(b), with the addition of only one square in g2, Black is sure to lose all of the northern edge, part of the eastern edge, as well as the game. If the edge table evaluator only considers edge discs, it would not realize the danger of this position.

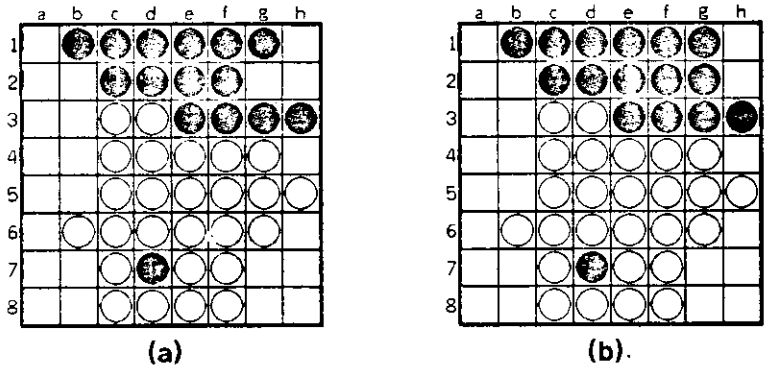


Figure 3-4: An example to illustrate the necessity to include the X-square in the edge table evaluation - (From the Waterloo Tournament: BILL B-61 - CASSIO W-3).

This might lead one to believe that X-squares are always bad, and that X-square moves should never be made. However, this is not the case. Figure 3-5(a) shows an example, in which BILL made the excellent x-square move b2. This move apparently gives up not only the northwestern corner, but also the entire

northern edge. However, a number of moves later, BILL was able to gain control of both the eastern and the western edge 8 moves later. In addition to corner/edge sacrifices, sometimes an x-square move does not relinquish the corresponding corner but captures it! In Figure 3-5(b), BILL moved to b7, which provides only three moves to the opponent - a6, a7, and b8. All three moves flip the b7 x-square back to White, thus giving the a8 corner to BILL. Therefore, although x-square moves are often very poor moves, they are sometimes acceptable or even brilliant moves.

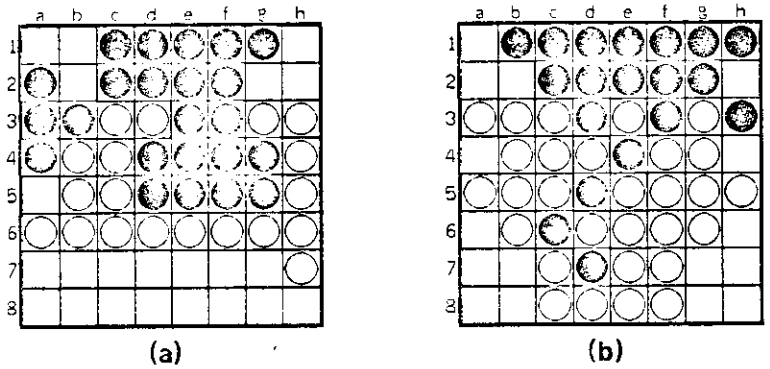


Figure 3-5: (a) shows the position before BILL (Black) moves to b2 (BILL B-44 vs. ALDARON W-20). (b) shows the position before BILL (Black) moves to b7 (BILL B-63 vs. CASSIO W-3). Both examples illustrate that X-square moves can sometimes be excellent moves. In (a), Black sacrifices the northern edge to gain possession of the western and eastern edges 8 moves later. In (b), White is forced to flip the b7 discs back to White, and thereby losing the corner.

Therefore, for each edge, we must consider the eight squares on the edge, as well as the two adjacent x-squares. This results in a total of 3^{10}, or 59049, combinations. The total edge advantage term is the sum of the four slightly overlapping evaluations. Since certain edge value depends crucially upon whose turn it is, it is necessary to have separate values for Black or White to move. In order to save storage, we store only values for Black to move. A simple color-reversal is executed for White's move.

We now introduce the following edge table generation algorithm. This algorithm consists of two passes of all edge configurations:

1. Static value initialization.
2. Probabilistic minimax search.

3.2.1.1 Static Value Initialization

Before the search for most suitable value for each of the 59049 positions could take place, each position is first assigned some static value, which represents the goodness of the position without consideration of what might happen later.

The first stage of the initialization assigns a value to each of the edge discs for each player. This value is determined by considering two factors, namely, square type and disc stability. The static values used are shown in Table 3-3. An *unstable* disc is one that can be flipped the next turn. All discs (Black and White) in Figure 3-6(a) are unstable. A *stable-3* disc is a disc that can *never* be flipped no matter what happens. The first four discs in Figure 3-6(b) and all discs in Figure 3-6(c) are of this type. A *stable-1* disc is a stable disc that can be flipped if and only if the side owning this disc is forced into playing to a particular square. For example, if Black were forced to go to square 6 in Figure 3-6d, it would no longer have stable discs. These two Black discs are of type *stable-1*. A *stable-2* disc is similar, except one forced move by each side is necessary to flip it. The two adjacent Black discs in Figure 3-6e are examples of this. An *unanchored stable* disc is between two sequences of opponent's *unstable* discs, such as the two White discs in Figure 3-6f. An *alone* disc is one that is next to a blank on each side, as shown in Figure 3-6g. A *semi-stable* disc is one that does not belong in any of the above categories, as shown in Figure 3-6h.

	Corner	C-Square	A-Square	B-Square
Unstable	———	-50	20	15
Alone	———	-75	-25	-50
Semi-stable	———	-125	100	100
Unanchored Stable	———	———	300	200
Stable-1	———	800	800	800
Stable-2	———	1000	1000	1000
Stable-3	800	1200	1000	1000

Table 3-3: The static values used to initialize edge values before the probabilistic minimax search.

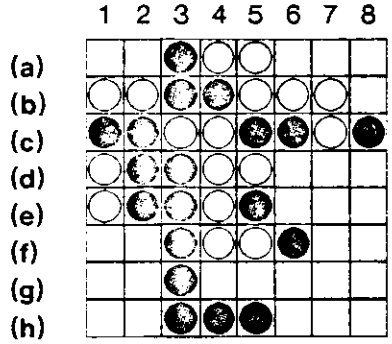


Figure 3-6: Some examples of stability types of the edges.

Each position is assigned a static value for Black by adding the value for each Black disc, and subtracting the value of each White disc. Next, the X-squares are evaluated, and small values are added or subtracted for the goodness of the X-square positions for Black. The resulting value for each position is a static measure of goodness for Black.

The static value is accurate in estimating many edge configurations, but it may be misleading in others. Consider the southern edge in Figure 3-7. If White could play into the *hole* e8, he could possess most of the southern edge. Furthermore, it is extremely likely that d7, e7, or f7 is Black because c8, d8, and f8 are all occupied by Black. This, in turn, increases the likelihood that White could move into e8. Therefore, this edge should have an extremely low value for Black. However, according to the values in Table 3-3, this edge evaluates to -100 on a scale of -8000 to 8000. Furthermore, this position is certainly worse for Black if it were White's move. Yet, the static table contains the same value with either player to play. Therefore, it is necessary for the edge generation algorithm to perform search and consider the probability of the relevant events.

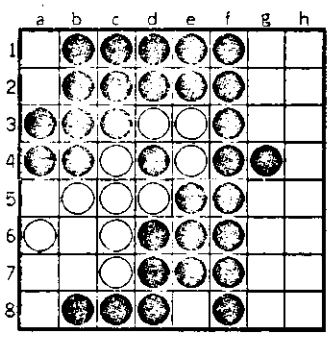


Figure 3-7: This position (BILL W-60 vs. GRAY BLITZ B-4) illustrates the inadequacy of static edge evaluation. The southern edge is extremely dangerous for Black, yet it evaluates to almost even.

3.2.1.2 Probabilistic Minimax Search

After the generation of the static values, a variant of the minimax search algorithm is applied to all possible edge configurations. The outline of the minimax search is as follows:

1. All completely filled positions have accurate static values, so they are marked as having *converged.* All other positions are marked as *not converged.*
2. Fill in the *not-converged* positions for each color to move by recursively computing the value for the positions after:
    a. Each move by the color that flips edge discs. (A *legal* edge move)
    b. Each move by the color that flips no edge discs. (A *possible* edge move)

After all recursive calls return, these values of these children nodes are negated and combined into the value for the parent position (the combining algorithm will be described later). The parent position is marked as having *converged.*

3. The above procedure is called with the empty edge, which will recursively fill in values for all the other positions.

One problem with this algorithm is that it assumes each player is forced to choose among the moves on each edge, which is far from reality. Instead, for each position with a player to move, that player has the option of passing. Pass is considered a legal move which makes no changes on the edge, and gives the other player the right to move[8].

To complete the algorithm, we now describe how the children values are backed up to form the parent value. Clearly, the standard minimax algorithm which takes the best move does not apply because the *possible* moves may be illegal. Furthermore, some *possible* moves are more likely than others. For example, a move into a *hole* of the southern edge of Figure 3-7 is much more likely than a move into a corner in the same edge.

In order to properly combine these values, we associate each move to an edge with a probability. These probabilities are fabricated based on a number of factors, including:

1. A *legal* move has probability 1.0.
2. A move to a corner is very unlikely unless the corresponding x-square is occupied.
3. A move to any square is
    a. more likely if there are opponent's discs nearby on the edge.
    b. less likely if there are our discs nearby on the edge.
    c. more likely if the edge has more discs.

The above and a few other minor factors are tuned until a number of selected edge positions are all assigned reasonable values.

To combine the probabilities and scores of all the children of an edge position, we introduce the following probability-combining algorithm:

1. Find the best *legal* move, L, with probability 1.0, and score $S(L)$. All other *legal* moves can be ignored because L is *always* a better move.
2. Initialize the value of the edge to 0, and remaining-probability, or R to 1.0.

---

[8] This actually introduces an infinite loop, and had to be dealt with by a complex variation of the probability-combining algorithm that considers the static value of the *not-converged* positions.

3. Sort all *possible* moves, and loop through them from best to worst: For each *possible* move M_i, with probability $P(M_i)$, and score $S(M_i)$:

   a. If M_i is worse than L, quit the loop.

   b. Otherwise, increment the value of the edge by $P(M_i) \times R \times S(M_i)$.

   c. Decrement R by $P(M_i)$.

4. Increment the value of the edge by $R \times S(L)$.

Figure 3-8(a) shows an example where Black has three *legal* moves (to squares 3, 7, and NOMOVE), and three *possible* moves (to squares 1, 8, and 10). Figure 3-8(b) shows the scores[9] and probabilities of each move as backed up by searching. The value of this position with Black to move is computed as follows: 92% of the time, Black will be able to move to square 1 (the best *possible* move), obtaining a partial score of $450 \times 0.92 = 414$. In the event that move 1 is illegal (8%), Black would move to square 8 (the second best *possible* move) 2% of the time, for a partial score of $400 \times 0.02 \times 0.08 = 0.64$. The final *possible* move (square 10) is inferior to the best *legal* move (NOMOVE), so it is not considered because *even if the move to square 10 were legal, we are better off making NOMOVE.* So NOMOVE is the course of action in the event that neither of the best two *possible* moves is legal (8% × 98%, or 7.84%) of the summation is $200 \times 0.0784 = 15.68$. Therefore, the evaluation of the position shown is $414 + 0.63 + 15.67 = 430.30$.

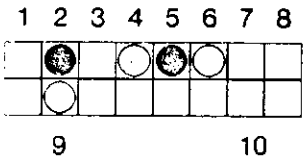
BILL's edge table generation algorithm requires about 5 CPU minutes. Since the authors are not Othello experts, it is difficult to judge the correctness of the edge values; however, BILL has never lost a game or an edge due to a lack of understanding of edge configurations.

3.2.2 Internal Tables

As stated in Section 2.2, there are a number of non-edge features important in the construction of an evaluation function:

- **Mobility.**
  - How many moves each player has.
  - Where each move is.
  - How many discs each move flips.
- **Potential mobility** - or frontier discs.
  - How many empty squares are next to each player's discs.
  - How many of each player's discs are next to empty squares.

[9] These scores are in range -1000 to 1000, and not -8000 to 8000.



(a)

	Score	Prob
Legals moves		
Square 3	-600	1.00
7	-500	1.00
NO MOVE	200	1.00
Possible moves		
Square 1	450	0.92
8	400	0.02
10	-100	0.80

(b)

Figure 3-8: A node in the edge table generation. Numbered squares are part of the edge table; unnumbered squares need not be empty. The table shows the values returned by recursive probabilistic minimax for the *legal* and *possible* moves.

- **Sequence and square value.**
  - The length of each sequence of discs.
  - The location of each sequence.
  - Occupying squares in central and surrounded positions.

But an evaluation function that processes the entire board to search for these features is prohibitively expensive. BILL exploits the fact that *all of these features can be measured or approximated by examining relationships between sequences of adjacent squares.* Each sequence of adjacent squares must be on one of the lines. Therefore, instead of looking at a large number of adjacent squares, we could capture all of these ideas in a pre-compiled table of *lines* on the Othello board.

BILL has a total of:

- 3 6561-entry orthogonal tables.
- 1 6561-entry 8-diagonal table.
- 1 2187-entry 7-diagonal table.
- 1 729-entry 6-diagonal table.

- 1 243-entry 5-diagonal table.
- 1 81-entry 4-diagonal table.
- 1 9-entry 3-diagonal table.

Because of the symmetry of the board, each table is used to evaluate several (2 for the 8-diagonal table, and 4 for all other tables) different lines on the board. Thus, the evaluation of internal advantage term is a summation of 34 table look-up's.

The edge table was generated with the probabilistic minimax search because edge discs could only be flipped with another edge move. Such is not true with internal discs. Moreover, the features mentioned above can be computed easily (but is too expensive to do during run-time). Therefore, it was decided to generate the internal tables with a much simpler algorithm. We will now discuss how the internal tables are precomputed to capture each of the three abovementioned major features.

Although all three types of knowledge are compressed into one 32-bit table, the occupied-squares component requires different coefficients, so it is assigned the most significant 16 bits of each table entry. Mobility and potential mobility are closely related, and are combined into the least significant 16 bits of each table entry. Thus, even though there is only one set of internal tables, the occupied-squares values are conceptually in a table different from that of the mobility values.

3.2.2.1 Weighted Mobility

Because mobility is not just a simple move count, BILL tries to evaluate moves by assigning higher values to moves that are likely to be desirable. Since each line is evaluated independently, BILL cannot know with certainty that a move is good or bad. However, this provides an excellent estimate most of the time.

Weighted mobility is measured along each line by using two sources of information. The first is *where* the move is to. For example, a move to an A-square that flips the X-square is likely to be inferior. The second type of information is how many discs are flipped. In accord with the strategies discussed in Section 2.2, moves that flip many opponent's discs are considered inferior, and appropriate values are assigned.

Because of the dominant importance of current mobility, this term is weighed more heavily than potential mobility.

3.2.2.2 Weighted Potential Mobility

Because mobility is so crucial to good Othello play, it is important to play into positions that are likely to yield moves in the future. Potential mobility measures this likelihood. This measure can be captured by examining adjacency between empty squares and pieces, or counting *frontier discs*. Rosenbloom [2] combined

three counting methods in IAGO. BILL uses two measures similar to Rosenbloom's ideas, but also weighs potential moves according to desirability.

For each possible line configuration, the following simple algorithm is executed to compute potential mobility: For each empty square, look at its neighbors, and subtract points for the color neighboring it. For each Black disc, if it is next to an empty square, subtract points for Black, and do the same for White. Discs on the edge are not counted because they are not truly frontier discs, and cannot provide the other side with moves.

Similar to mobility measurement, the number of points to subtract is weighed according to how good the potential move is. If the potential move is to a central location, it is worth more points; if the potential move is to an x-square, or would flip an x-square, it is worth very few points.

For each line-configuration, the total weighted potential mobility value is added to the total weighted mobility. After normalization, the weight for potential mobility is considerably smaller than weighted mobility.

3.2.2.3 Occupied-Squares

The occupied-squares component of the evaluation function consists of two separate knowledge sources, namely, sequence penalty and weighted square values.

At almost all times, it is a good idea to avoid long sequences of one's discs. This long sequence may be a wall, and it increases one's disc count. BILL recognizes all long sequences in each line, and penalizes the player owning that sequence. The penalty depends on the location of the sequence and the number of discs in it.

Another component in this table is a weighted square measure. Many misinformed programs and players use only a weighted square evaluation function. While that is clearly inadequate by itself, a weighted sum of a player's squares is of some value. Each player's discs are weighed according to the following rules:

1. Discs between other player's discs are very good, especially when there are real moves.
2. Discs that can neither flip nor be flipped are bad (likely walls).
3. Centrally located discs are better than peripherally located discs.

Sequence and weighted square values are summed as the occupied-squares advantage term in the evaluation function. Both measures are very good approximations in the beginning of the game; however, during the midgame, mobility is much more important, and during the endgame, occupied-squares

information undermines the ultimate goal of maximizing stable sequences and discs. Therefore, its weight in the evaluation function drops drastically towards the end of the game.

3.2.3 Disc Count

The game of Othello was so named because of its unpredictability [11]. With one move, a maximum of 19 discs can be flipped. Thus, it is possible for one player to be ahead with the score of 50-13 before the last move, yet lose the game. In the late-opening and mid-game, most experts are convinced that it is best to keep a minimal disc count [4].

Although it is usually best to keep a minimal disc count, this particular strategy is an indirect and crude approximation of the mobility optimization strategy. Thus, BILL does not consider disc count as an addition source of information for non-terminal positions, except in cases of a wipe-out (see below). However, for the terminal positions (in the end-game search), BILL relies upon disc count exclusively.

Throughout the game, a disc count for each player is maintained incrementally, and is used to evaluate terminated positions in the endgame search.

3.2.4 Wipe-out Avoidance

The rules of Othello stipulate that if one player has no discs left he has lost the game with a score of 64-0 whether or not the board is completely filled. This necessitates a wipe-out avoidance measure in the evaluation function. Consider a board position where Black wins by wiping out White. Without a wipe-out avoidance, when this position is generated in a non-endgame search tree, its evaluation will be highly positive for White because of the Black sequences (walls) and frontier discs.

Therefore, the evaluation function first checks whether the disc count for one side is zero. If so, that side gets the evaluation of $-\infty$. Since the disc count is maintained incrementally, this measure is a very inexpensive insurance against deep-searching maximum-disc opponents.

3.3 Other Features

3.3.1 The Opening Book

Due to the difficulty most human and computer Othello players have in the opening (BILL is no exception), it was decided that BILL needed an extensive opening book. Another reason for creating a deep opening book was to give BILL a time advantage by having it respond instantly for the first few moves. The importance of an opening book is demonstrated by the success of ALDARON. ALDARON's opening book was

generated by its author, who is an Othello expert. The superiority of ALDARON's opening book is one of the reasons for BILL's defeat in the North American Computer Othello Championship.

Originally, BILL had a small opening book of 180 positions based on Othello literature. This was felt to be inadequate, and human playing style in the opening was found to be incompatible with subsequent moves by BILL. By applying the algorithm to be described, BILL automatically generated an opening book contained a total of 14400 positions.

3.3.1.1 The Structure of the Book

The structure described below was chosen to minimize space requirements. First of all, each of the four initial moves are equivalent. The book was only generated for one of them (e6) and a rotation is performed if the actual game starts with a move other than e6. Secondly, there is one book for when BILL plays White and another for when BILL plays Black. Each book is a tree that contains lines of play that branch out for the opponents choice but only have the one response for the color whose book it is. To make this more concrete, consider the White book. This book is a tree with the following properties:

- Each position with *Black* to move has a small number of descendants, usually from 2-5. The descendants saved are felt to be the best choices for Black.
- Each position with White to move has only one choice, which is felt to be the best possible move by White from that position.

Thus, only the even levels (Black to move) increase the size of the tree. This lowering of the branching factor doubles the effective depth of the book.

3.3.1.2 Method of Generation

To generate the book, a version of BILL modified for generation of the opening book was provided with the terminal positions in BILL's old Black and White opening books. Each position was expanded and the sub-trees were combined to get a Black book and a White book. To decide how much effort to invest in each subtree, a parameter called the a *use count* is assigned to the beginning node. The use count[10] is used to control the exponential tree growth. To expand a node for White, the following algorithm is used:

- If it is White to move, determine the best move using BILL's searching techniques. The use count of the one descendant is equal to the current node's use count. In other words, there is no penalty for expanding this type of node because it adds nothing to the number of root nodes. The only descendant is then recursively expanded.
- If it is Black to move, each of his options are searched to determine its value. The current use count, along with the backed-up evaluations of the various choices, governs how many choices will be saved and expanded:

[10] Modeled after the description in [12].

1. The nodes are sorted in descending order of preference to Black.
2. A use count is computed for each descendant:
   a. The first child is assigned a use count of *CurrentCount* - 10.
   b. FOR i = 1 TO *NumberOfDescendants*

      $$UseCount[i] = UseCount[i-1] + DropFunction(eval(i) - eval(i-1))$$

      where the drop function considers the difference in evaluations and returns a penalty. This method ensures that each node receives a penalty in roughly inverse proportion to its relative worth.
3. All the nodes with negative use counts are terminated.

Each of the saved descendants (those with a non-negative use count) is then recursively expanded.

3.3.1.3 Results

The size of the final books were as follows:

- For Black: 8000 positions, 2000 lines of play, maximum depth of 22 plies
- For White: 6400 positions, 1700 lines of play, maximum depth of 26 plies

The usefulness of the book was analyzed by having BILL play itself repeatedly, with varying time allocations, having one side use the book and the other not. The side using the book had an average margin of victory 8 discs greater than the side without the book. Based on this result, it can be concluded that the book was useful.

One interesting result that grew out of generating the book was the conjecture that Black has a win from the initial position. This conclusion is based on the fact that most of the lines of play in both books resulted in positions where Black had the advantage. This tendency was most noticeable in the White book. Further work may be carried out to either strengthen or disprove this conjecture.

3.3.2 Think Ahead

One feature of BILL that proved extremely useful was *think ahead.* That is, BILL "thinks" on its opponent's time. This utilizes time that would otherwise be wasted waiting for the opponent's move.

To accomplish think ahead, BILL makes a guess as to what move the opponent will make by either looking in the hash table or by performing a search to a preselected, small search depth. BILL makes the guessed move for the opponent and proceeds to do a normal zero-window search. This maximum depth of this search is pre-determined by the opponent's Othello rating (an input to BILL) and the time allocated for the whole game. The larger the time allocation, or the stronger the opponent, the deeper the maximum depth. If BILL completes the search before the opponent completes his or her move, then BILL chooses

another move to search by looking in the killer table. In the most unlikely event that all moves have been searched to the maximum depth, the maximum depth is incremented by one. This search and re-search process continues until the opponent responds.

If one of the guessed moves was correct, BILL has saved however long was invested in searching the correct move. The savings are passed on by reducing the upper and lower time limits described in Section 3.1.3. If BILL's time allocation becomes negative, that means that BILL has exceed its time allocation at no cost to itself! If BILL guessed incorrectly, that time spent searching on the opponent's time was simply wasted. In the ideal case, BILL will spend all of its time searching the move that will be made.

This think-ahead scheme is somewhat different from the standard technique of selecting one move for the opponent and assuming that he makes that move. In actual tournament-condition games against good opponents, BILL sets the maximum depth sufficiently high that it rarely attempts to guess a second move for the opponent. However, against weaker opponents, BILL is unlikely to guess correctly. Furthermore, in a quick game where each side has only a few minutes for the entire game, additional opponent moves should be investigated if the opponent spends too much time thinking. In these two cases, BILL's strategy is superior to the standard technique.

Think ahead has proved extremely useful, its utility increasing with the strength of opposition. At the Waterloo Othello Tournament, BILL guessed 27% of its opponent's moves correctly, while the corresponding figure for the NA Championship is 51%. Against IAGO, BILL benefits even more from think ahead. Because IAGO does not have think-ahead, BILL can actually defeat IAGO with only one-fifth the time allocation of IAGO.

4. Results

The first version of BILL was written in the summer of 1985 by six high school students as a computer science team project for Pennsylvania Governor's School for the Sciences (PGSS) under the first author's supervision [6]. This version was written in Common LISP for the IBM-PC and the Perq. It employed a simple alpha-beta search and used an evaluation function composed of move-counting and simple edge and corner tables. Because of the enormous expense of move counting and the slow speed of LISP, this version could only search 3 levels. In spite of these shortcomings, this version defeated all but one of its human opponents (Scott Craig, a very experienced Othello player), and all but one of its computer opponents (IAGO, the 1981 world champion). In a demonstration to Governor Thornburg of Pennsylvania, BILL convincingly defeated its opponent.

This PGSS project was inspired by Paul Rosenbloom's IAGO [2] (see Appendix I). Because of IAGO's availability at CMU and its outstanding ability, BILL's progress was often measured by playing against IAGO. The PGSS version of BILL [6] was a very good Othello player; however, it was soundly defeated by IAGO. Since that version of BILL is basically a simplified version of IAGO, and it ran in a much slower environment, this defeat was expected.

After PGSS ended, BILL was rewritten by the authors in C on a VAX-11/785, which increased its speed by a factor of 10, and occasionally it was able to defeat IAGO. Then, a number of useful features such as the linked-move killer table, the hash table, the zero-window search, and the two-phase endgame search were added. This change led to a version somewhat better than IAGO.

Finally, think-ahead was added and modifications to the evaluation function were made, including the extensive use of tables. With its new evaluation function, BILL was able to defeat IAGO consistently, although there were many close games.

This version of BILL was entered in the Waterloo Computer Othello Tournament on November 9, 1985. This tournament consisted of 10 programs, most of which were from Canada. BILL won all four games with very large margins, and captured first place. These games are listed in Appendix II as Figures II-1 to II-4.

After the Waterloo Tournament in November 1985, the evaluation function was modified into its present three-component form (previously, the *mobility* and the *occupied-squares* terms were combined). Moreover, the opening book was generated. This version of BILL is able defeated IAGO by much more convincing margins; two examples of BILL's playing strength can be seen in Figures II-14 and II-15.

This was the version entered in the North American Computer Othello Championship on February 9, 1986. 11 programs were entered in this tournament. BILL won 7 games out of 8, placing second after Charlie Heath's ALDARON, which accumulated 7 wins and one draw. BILL's only loss was to ALDARON. This loss was due to the color BILL drew in that game. Recall that we conjectured that White is at a disadvantage in Section 3.3.1. and BILL unfortunately drew White against ALDARON. In an unofficial rematch with the colors reversed, BILL defeated ALDARON. This is additional evidence for our conjecture.[11] Furthermore, BILL also defeated XOANNON, the only program that did not lose to ALDARON during that tournament. The eight tournament games and the unofficial game against ALDARON are listed in Appendix II as Figures II-5 to II-13.

From the above record of BILL's performance, it is clear that BILL is one of the best Othello computer programs. It would be interesting to compare BILL's ability to that of human experts; however, human players prohibited computer entries in human tournaments. Nevertheless, since IAGO was believed to be as good as, if not better than, the best human players [2], it is likely that BILL is a better Othello player than any human expert.

[11] This is not to say, however, that Black wins all, or even an overwhelming number of the games in Othello tournaments. This preference of Black is only noticeable at very high level of play.

5. Future work

Although BILL is a very strong player, it has some remaining problems that are areas for continuing work. Many of these problems involve tuning the tables used in BILL's evaluation function. The tables were constructed based on the authors' Othello intuition, which is not the most reliable source, since neither author is an expert player. Expert opinion and advice are currently being sought.

One problem of the table-based approach is the inaccurate measurement of mobility due to the inability to capture the interaction of internal tables. In some cases, the mobility term is not as accurate as actually counting the moves would be. For example, if a move flips one disc in each of the eight directions, the table approach will conclude that it is an acceptable move because each table is only aware of the flip of two discs. In reality, such is move is quite poor. In order to measure mobility accurately, it is necessary to add a non-table-based evaluation component. This would be an important improvement if efficient algorithms can be developed.

A minor problem is that the edge probabilities in the minimax search were fabricated. It was felt that estimated probabilities from a large number of random games would improve the edge table.

It is not clear that the assumption of linearity in the evaluation is valid. A new non-linear evaluation function is currently being generated with the aid of pattern classification techniques. It is hoped that this classical approach is applicable to our domain.

Finally, BILL still has the difficulty in the opening, in spite of its enormous book. The inadequacy of the evaluation function in the opening stages of the game is responsible for this. Since all Othello programs are weakest in their opening, solution to this problem may be elusive.

Current work is in progress to remedy all of the above-mentioned weaknesses of BILL. With an expert-tuned non-linear evaluation function, many of these difficulties will hopefully be eliminated or alleviated.

6. Conclusion

While successful Othello-playing programs have been created before, most have been impaled on the twin horns of the search-knowledge dilemma. Previously, if a program relied on searching but had a fast (and therefore limited) evaluation function, it would lose to human experts through lack of understanding. Similarly, if a program relied on knowledge and neglected searching, it may lose from lack of vision. BILL avoids both problems by being efficient yet knowledge-intensive.

BILL's evaluation function captures many of the important strategic concepts in Othello such as mobility and edge control, yet is extremely efficient through its use of table look-up. The use of tables shifts most of the computational burden from BILL's evaluation function to table-generation programs. BILL's ability to examine 1100 nodes per second is indicative of this. BILL's success can also be attributed to state-of-the-art artificial intelligence techniques used, such as a zero-window search, hash table, a linked-move killer table, and a two-phase end-game search.

We demonstrated the usefulness of these techniques with a number of controlled experiments, and the measured searching statistics of BILL. BILL's performance against other strong programs is additional evidence supporting the utility of these techniques, particularly that of a table-based evaluation function. It is hoped that future game-playing or searching programs will be able to make use of some of the concepts described in this paper.

Acknowledgments

The authors wish to thank Scott Craig, George Postelthwait, Steve Racunas, Andy Serotta, and George Wadsworth, the other members of the PGSS team that wrote the first version of BILL; Hans Berliner for his advice and suggestions; Paul Rosenbloom for writing IAGO, a patient teacher and worthy opponent for BILL; Raj Reddy for his support and encouragement; and Hans Berliner and Gordon Goetsch for reading drafts of this paper.

I. Comparison Between IAGO and BILL

IAGO, the winner of the 1981 North American Othello Championship, was among the first to demonstrate the plausibility of creating a computer game-playing program that is equal or superior to the best human players. It was IAGO's performance that encouraged the proposal of an Othello project at the Pennsylvania Governor's School for the Sciences.

There are two reasons to compare BILL with IAGO. Firstly, a comparison of techniques will identify which techniques of BILL are influenced by IAGO, and which ones are original. Secondly, a comparison of performance will suggest to what extent the ideas presented in this paper have been useful to BILL.

I.1 A Brief Description of IAGO

IAGO uses an iteratively-deepened alpha-beta search and a two-phase endgame search. To aid the search, IAGO maintains *response killers*, which is similar to the killer algorithm in BILL, but is not as versatile or efficient. Instead of a hash table, it saves the first 3-ply of the tree. IAGO's timing algorithm is also not as sophisticated as BILL's. It attempts to estimate the time to complete another level of search, instead of using an alarm as do BILL and Hitech [7].

The greatest problem of IAGO is the speed of its evaluation function. While its measurement of mobility is sometimes more accurate than BILL's, it required many expensive operations. As a result, it could only examine 150 nodes per second while BILL could examine 1100. This deficiency, coupled with its use of less sophisticated ordering technique, resulted in the loss of almost two levels of search, which is detrimental in a game that relies so heavily on searching.

I.2 A Tabular Comparison Between IAGO and BILL

Table I-1: Comparison between IAGO and BILL[12,13]

	Iago	Bill
Search	$\alpha\beta$ with iterative deepening	$\alpha\beta$ with zero-window, iterative deepening, and forward pruning
Ordering Optimization	Killer Response Table Keeps 3-ply tree	Hash Table Linked-move Killer Table
Repetition Avoidance	Keeps 3-ply tree	Hash Table
Evaluation	Edge Table Internal Stability Current Mobility Potential Mobility	Edge Table Current Mobility Table Potential Mobility Table Occupied-Square Table
Table Generation	Piece removal with fabricated probabilities	Probabilistic minimax with fabricated probabilities
Opening Book	Small, Manual (< 5 plies; 20 positions)	Automatic Generation (< 27 plies; 14,400 positions)
Timing	Dynamic allocation	Dynamic windowed allocation with alarm
I/O	Waits for opponent	Thinks on opponent's time by updating the hash table.
Hardware	DEC 2060 (CMU-CS-C)	VAX/11-785 (CMU-CS-SPEECH2)
Language	SAIL	C
Nodes/Sec	150	1100
Branching factor	4.0	3.7
Levels Searched	6.3	8.0

[12] Branching factor is computed assuming self play and a branching factor of 10.0 with a full minimax search.

[13] Levels searched is computed under tournament conditions.

II. Transcripts of Bill's Games

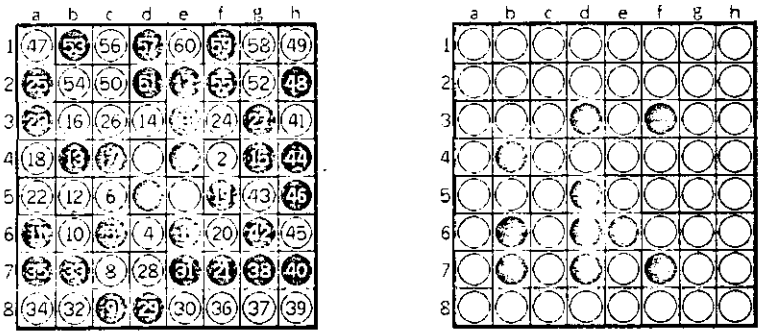


Figure II-1: Waterloo Othello Tournament (OTHELLO^0 B-10 vs. BILL W-54).

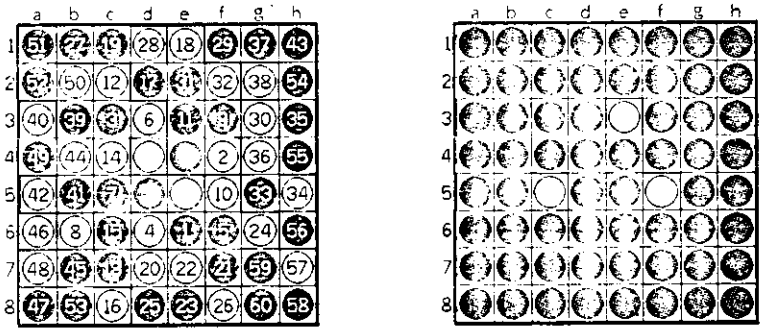


Figure II-2: Waterloo Othello Tournament (BILL B-61 vs. CASSIO W-3).

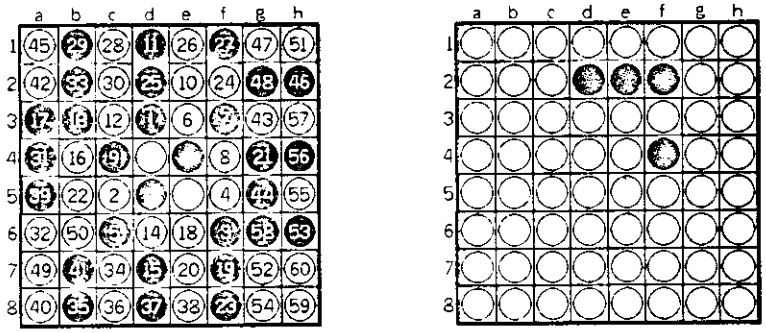


Figure II-3: Waterloo Othello Tournament (GRAY BLITZ B-4 vs. BILL W-60).

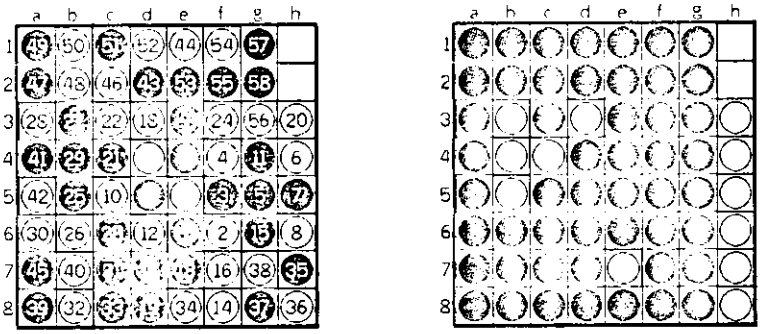


Figure II-4: Waterloo Othello Tournament (BILL B-50 vs. BARNEY W-12).

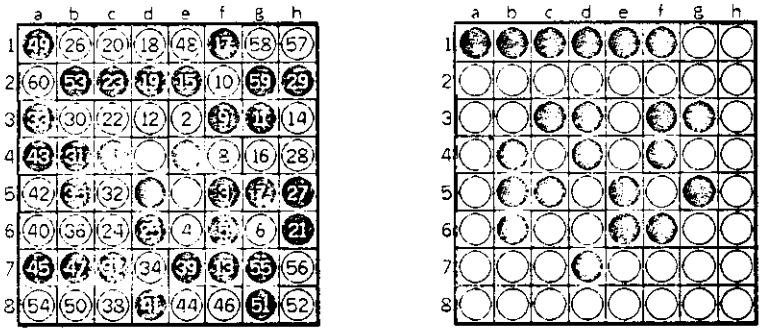


Figure II-5: NA Othello Championship (BRAND B-21 vs. BILL W-43).

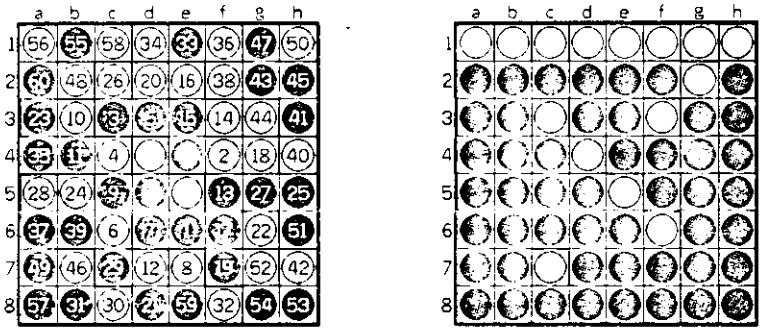


Figure II-6: NA Othello Championship (BILL B-49 vs. IPSC OTHELLO W-15).

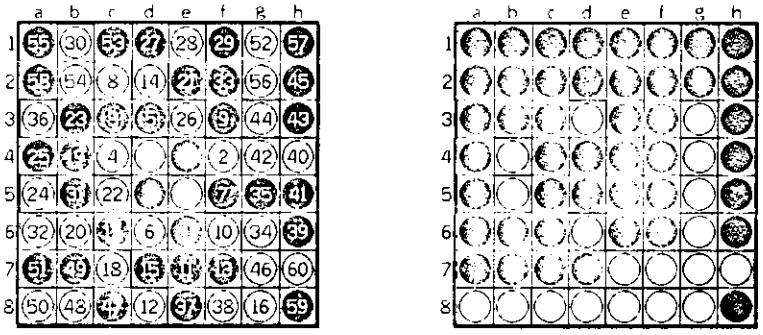


Figure II-7: NA Othello Championship (ALDARON B-45 vs. BILL W-19).

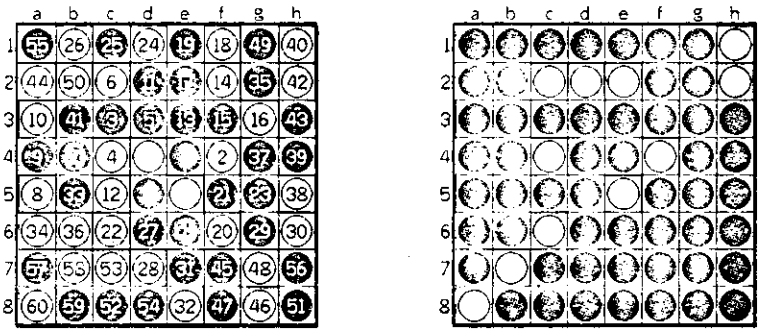


Figure II-8: NA Othello Championship (BILL B-53 vs. IAGO[Gupton] W-11).

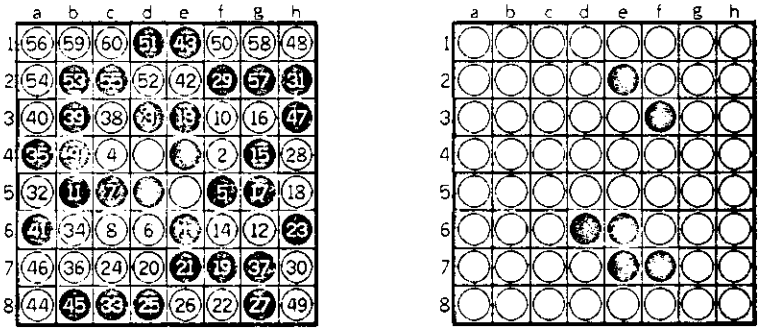


Figure II-9: NA Othello Championship (LGO B-6 vs. BILL W-58).

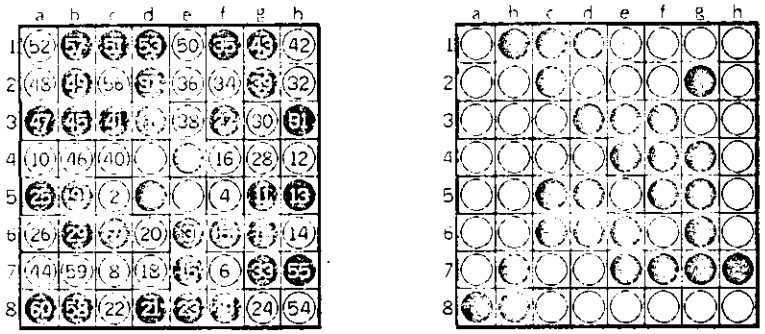


Figure II-10: NA Othello Championship (EXCALIBUR B-26 vs. BILL W-38).

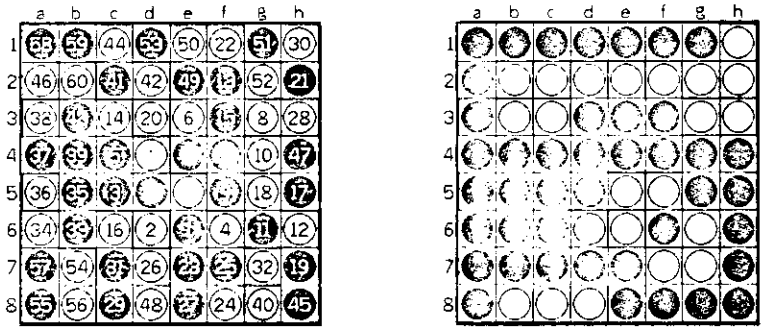


Figure II-11: NA Othello Championship (BILL B-42 vs. CUSTER W-22).

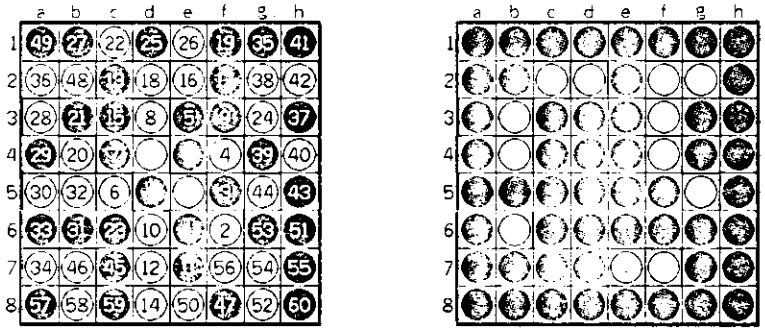


Figure II-12: NA Othello Championship (BILL B-52 vs. XOANNON W-12).

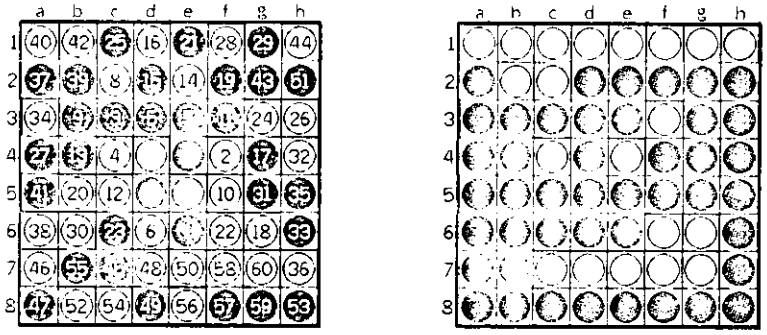


Figure II-13: Unofficial Game (BILL B-44 vs. ALDARON W-20).

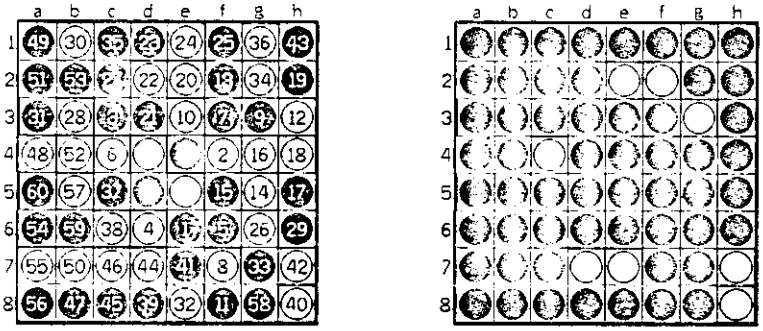


Figure II-14: Unofficial Game (BILL B-56 vs. IAGO W-8).

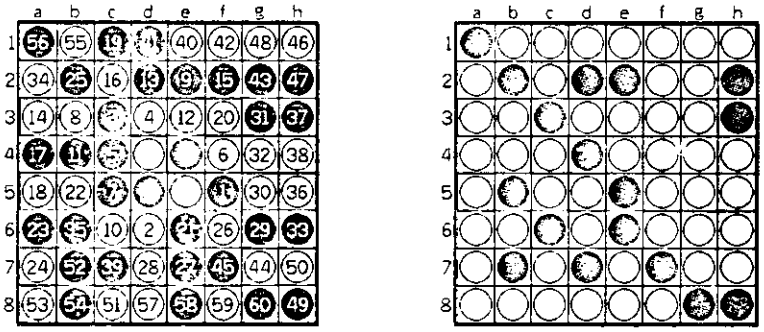


Figure II-15: Unofficial Game (IAGO B-17 vs. BILL W-47).

References

1. Weaver, D., "Black, White, and Gray", *Othello Quarterly*, Vol. 4, No. 2, Summer 1982, pp. 6-9.

2. Rosenbloom, P. S., "A World-Championship-Level Othello Program", *Artificial Intelligence*, Vol. 19, No. 3, November 1982, pp. 279-319.

3. Kierulf, A., "Brand : an Othello program", in *Computer Game-playing: Theory and Practice*, Halsted Press, 1983, pp. 197-208.

4. Landau, T., "Othello: Brief and Basic", *Othello Quarterly*, Vol. 7, No. 1, Spring 1985, pp. 3-14.

5. Slate, D. J., Atkin, L. R., "CHESS 4.6 -- The Northwestern University Chess Program", in *Chess Skills in Man and Machine*, Springer-Verlag, 1977, pp. 101-107.

6. Craig, S., Mahajan, S., Postelthwait, G., Racunas, S., Serotta, A., Wadsworth, G., "Bill : An Othello Program", *Journal of the PGSS*, Vol. 4, No. 11985, pp. 135-142.

7. Berliner, H., "Personal Communications", .

8. Pearl, Judea. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company, 1984.

9. Slagle, J. R., Dixon, J. K., "Experiments With Some Programs That Search Game Trees", *Journal of the ACM*, Vol. 16, No. 2, April 1969, pp. 189-207.

10. Berliner, H., "On the Construction of Evaluation Functions for Large Domains", *Proceedings of IJCAI-79*, 1979, pp. 53-55.

11. Gardner, M., "Mathematical Games", *Scientific American*, Vol. 236, No. 4, April 1977, pp. 134.

12. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers. II", *IBM Journal*, No. 11, November 1967, pp. 601-617.

13. Frey, P. W., "The Santa Cruz Open Othello Tournament for Computers", *Byte*, Vol. 6, No. 7, July 1981, pp. 26-37.

14. Maggs, P. B., "Programming Strategies in the Game of Reversi", *Byte*, Vol. 4, No. 11, November 1979, pp. 66-79.

15. Frey, P. W., "Simulating Human Decision-Making on a Personal Computer", *Byte*, Vol. 5, No. 7, July 1980, pp. 56-72.

16. Heath, C., "Flowers for Aldaron", *Othello Quarterly*, Vol. 4, No. 2, Summer 1982, pp. 9-12.

17. Sullivan, G., "Machine vs. Machine", *Othello Quarterly*, Vol. 4, No. 2, Summer 1982, pp. 13-18.