A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers

Abstract: Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations.

Introduction

The studies reported here have been concerned with the programming of a digital computer to behave in a way which, if done by human beings or animals, would be described as involving the process of learning. While this is not the place to dwell on the importance of machine-learning procedures, or to discourse on the philosophical aspects,[1] there is obviously a very large amount of work, now done by people, which is quite trivial in its demands on the intellect but does, nevertheless, involve some learning. We have at our command computers with adequate data-handling ability and with sufficient computational speed to make use of machine-learning techniques, but our knowledge of the basic principles of these techniques is still rudimentary. Lacking such knowledge, it is necessary to specify methods of problem solution in minute and exact detail, a time-consuming and costly procedure. Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort.

• *General methods of approach*

At the outset it might be well to distinguish sharply between two general approaches to the problem of machine learning. One method, which might be called the *Neural-Net Approach,* deals with the possibility of inducing learned behavior into a randomly connected switching net (or its simulation on a digital computer) as a result of a reward-and-punishment routine. A second, and much more efficient approach, is to produce the equivalent of a highly organized network which has been designed to learn only certain specific things. The first method should lead to the development of general-purpose learning machines. A comparison between the size of the switching nets that can be reasonably constructed or simulated at the present time and the size of the neural nets used by animals, suggests that we have a long way to go before we obtain practical devices.[2] The second procedure requires reprogramming for each new application, but it is capable of realization at the present time. The experiments to be described here were based on this second approach.

• *Choice of problem*

For some years the writer has devoted his spare time to the subject of machine learning and has concentrated on the development of learning procedures as applied to games.[3] A game provides a convenient vehicle for such study as contrasted with a problem taken from life, since many of the complications of detail are removed. Checkers, rather than chess,[4–7] was chosen because the simplicity of its rules permits greater emphasis to be placed on learning techniques. Regardless of the relative merits of the two games as intellectual pastimes, it is fair to state that checkers contains all of the basic characteristics of an intellectual activity in which heuristic procedures and learning processes can play a major role and in which these processes can be evaluated.

Some of these characteristics might well be enumerated. They are:

(1) The activity must not be deterministic in the practical sense. There exists no known algorithm which will guarantee a win or a draw in checkers, and the complete

REPRINTED FROM IBM JOURNAL OF RESEARCH AND DEVELOPMENT, VOL. 3, NO. 3, 1959; ©1959, 2000

0018-8646 / 00 / $5.00 © 2000 IBM

explorations of every possible path through a checker game would involve perhaps 10^{40} choices of moves which, at 3 choices per millimicrosecond, would still take 10^{21} centuries to consider.

(2) A definite goal must exist—the winning of the game—and at least one criterion or intermediate goal must exist which has a bearing on the achievement of the final goal and for which the sign should be known. In checkers the goal is to deprive the opponent of the possibility of moving, and the dominant criterion is the number of pieces of each color on the board. The importance of having a known criterion will be discussed later.

(3) The rules of the activity must be definite and they should be known. Games satisfy this requirement. Unfortunately, many problems of economic importance do not. While in principle the determination of the rules can be a part of the learning process, this is a complication which might well be left until later.

(4) There should be a background of knowledge concerning the activity against which the learning progress can be tested.

(5) The activity should be one that is familiar to a substantial body of people so that the behavior of the program can be made understandable to them. The ability to have the program play against human opponents (or antagonists) adds spice to the study and, incidentally, provides a convincing demonstration for those who do not believe that machines can learn.

Having settled on the game of checkers for our learning studies, we must, of course, first program the computer to play legal checkers; that is, we must express the rules of the game in machine language and we must arrange for the mechanics of accepting an opponent's moves and of reporting the computer's moves, together with all pertinent data desired by the experimenter. The general methods for doing this were described by Shannon[8] in 1950 as applied to chess rather than checkers. The basic program used in these experiments is quite similar to the program described by Strachey[9] in 1952. The availability of a larger and faster machine (the IBM 704), coupled with many detailed changes in the programming procedure, leads to a fairly interesting game being played, even without any learning. The basic forms of the program will now be described.

The basic checker-playing program

The computer plays by looking ahead a few moves and by evaluating the resulting board positions much as a human player might do. Board positions are stored by sets of machine words, four words normally being used to represent any particular board position. Thirty-two bit positions (of the 36 available in an IBM 704 word) are, by convention, assigned to the 32 playing squares on the checkerboard, and pieces appearing on these squares are represented by 1's appearing in the assigned bit positions of the corresponding word. "Looking-ahead" is prepared for by computing all possible next moves, starting with a given board position. The indicated moves are explored in turn by producing new board-position records corresponding to the conditions after the move in question (the old board positions being saved to facilitate a return to the starting point) and the process can be repeated. This look-ahead procedure is carried several moves in advance, as illustrated in Fig. 1. The resulting board positions are then scored in terms of their relative value to the machine.

The standard method of scoring the resulting board positions has been in terms of a linear polynomial. A number of schemes of an abstract sort were tried for evaluating board positions without regard to the usual checker concepts, but none of these was successful.[10] One way of looking at the various terms in the scoring polynomial is that those terms with numerically small coefficients should measure criteria related as intermediate goals to the criteria measured by the larger terms. The achievement of these intermediate goals indicates that the machine is going in the right direction, such that the larger terms will eventually increase. If the program could look far enough ahead we need only ask, "Is the machine still in the game?"[11] Since it cannot look this far ahead in the usual situation, we must substitute something else, say the piece ratio, and let the machine continue the look-ahead until one side has gained a piece advantage. But even this is not always possible, so we have the program test to see if the machine has gained a positional advantage, et cetera. Numerical measures of these various properties of the board positions are then added together (each with an appropriate coefficient which defines its relative importance) to form the evaluation polynomial.

More specifically, as defined by the rules for checkers, the dominant scoring parameter is the inability for one side or the other to move.[12] Since this can occur but once in any game, it is tested for separately and is not included in the scoring polynomial as tabulated by the computer during play. The next parameter to be considered is the relative piece advantage. It is always assumed that it is to the machine's advantage to reduce the number of the opponent's pieces as compared to its own. A reversal of the sign of this term will, in fact, cause the program to play "give-away" checkers, and with learning it can only learn to play a better and better give-away game. Were the sign of this term not known by the programmer it could, of course, be determined by tests, but it must be fixed by the experimenter and, in effect, it is one of the instructions to the machine defining its task. The numerical computation of the piece advantage has been arranged in such a way as to account for the well-known property that it is usually to one's advantage to trade pieces when one is ahead and to avoid trades when behind. Furthermore, it is assumed that kings are more valuable than pieces, the relative weights assigned to them being three to two.[13] This ratio means that the program will trade three men for two kings, or two kings for three men, if by so doing it can obtain some positional advantage.

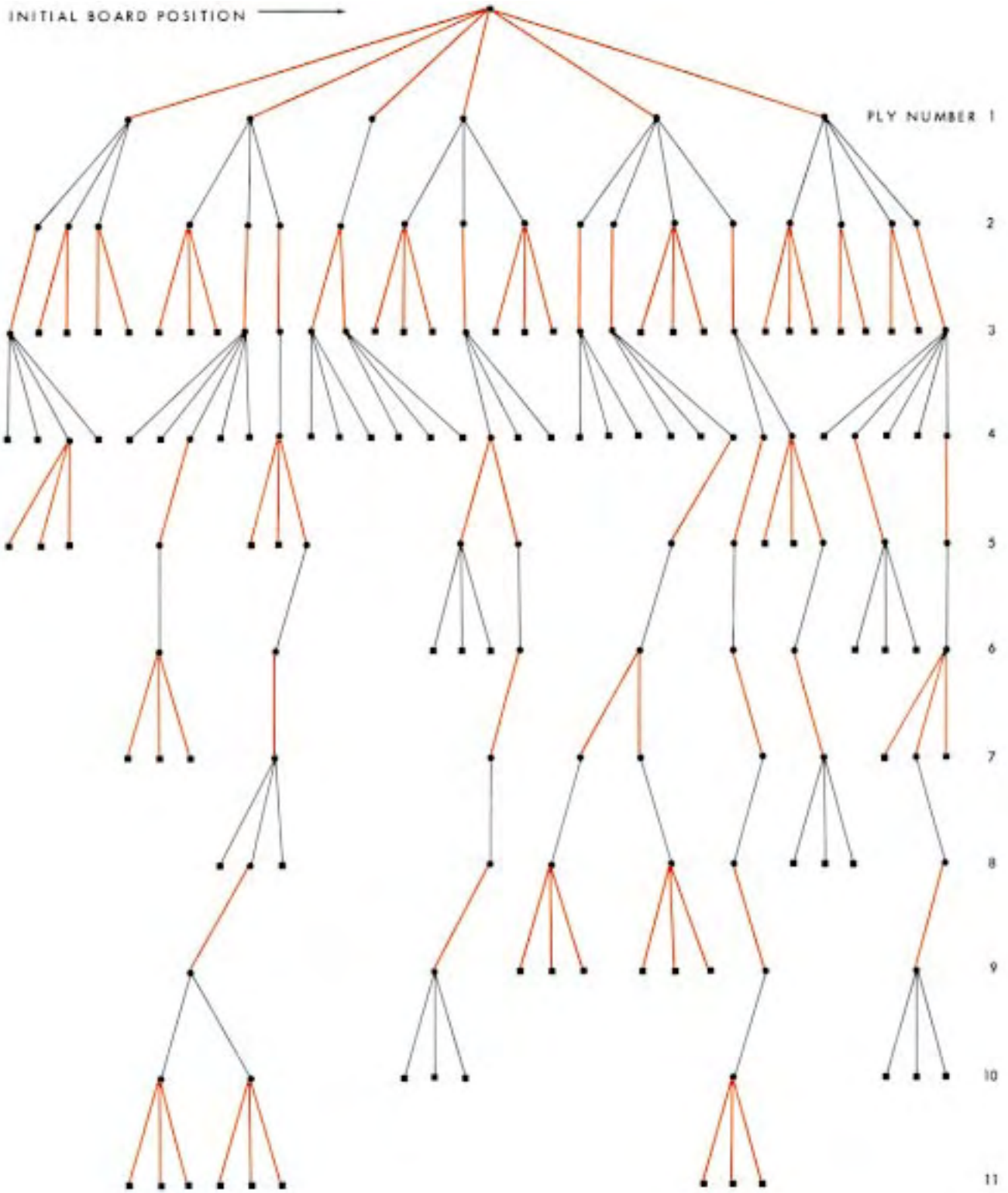


Figure 1 A "tree" of moves which might be investigated during the look-ahead procedure. The actual branchings are much more numerous than those shown, and the "tree" is apt to extend to as many as 20 levels.

The choice for the parameters to follow this first term of the scoring polynomial and their coefficients then becomes a matter of concern. Two courses are open—either the experimenter can decide what these subsequent terms are to be, or he can arrange for the program to make the selection. We will discuss the first case in some detail in connection with the rote-learning studies and leave for a later section the discussion of various program methods of selecting parameters and adjusting their coefficients.

It is not satisfactory to select the initial move which leads to the board position with the highest score, since to reach this position would require the cooperation of the opponent. Instead, an analysis must be made proceeding *backward* from the evaluated board positions through the "tree" of possible moves, each time with consideration of the intent of the side whose move is being examined, assuming that the opponent would always attempt to minimize the machine's score while the machine acts to maximize its score. At each branch point, then, the corresponding board position is given the score of the board position which would result from the most favorable move. Carrying this "minimax" procedure back to the starting point results in the selection of a "best move." The score of the board position at the end of the most likely chain is also brought back, and for learning purposes this score is now assigned to the present board position. This process is shown in Fig. 2. The best move is executed, reported on the console lights, and tabulated by the printer.

The opponent is then permitted to make his move, which can be communicated to the machine either by means of console switches or by means of punched cards. The computer verifies the legality of the opponent's move, rejecting[14] or accepting it, and the process is repeated. When the program can look ahead and predict a win, this fact is reported on the printer. Similarly, the program concedes when it sees that it is going to lose.

- *Ply limitations*

Playing-time considerations make it necessary to limit the look-ahead distance to some fairly small value. This distance is defined as the *ply* (a ply of 2 consisting of one proposed move by the machine and the anticipated reply by the opponent). The ply is not fixed but depends upon the dynamics of the situation, and it varies from move to move and from branch to branch during the move analysis. A great many schemes of adjusting the look-ahead distance have been tried at various times, some of them quite complicated. The most effective one, although quite detailed, is simple in concept and is as follows. The program always looks ahead a minimum distance, which for the opening game and without learning is usually set at three moves. At this minimum ply the program will evaluate the board position if none of the following conditions occurs: (1) the next move is a jump, (2) the last move was a jump, or (3) an exchange offer is possible. If any one of these conditions exists, the program continues looking ahead. At a ply of 4 the program will stop and evaluate the resulting board position if conditions (1) and (3) above are not met. At a ply of 5 or greater, the program stops the look-ahead whenever the next ply level does not offer a jump. At a ply of 11 or greater, the program will terminate the look-ahead, even if the next move is to be a jump, should one side at this time be ahead by more than two kings (to prevent the needless exploration of obviously losing or winning sequences). The program stops at a ply of 20 regardless of all conditions (since the memory space for the look-ahead moves is then exhausted) and an adjustment in score is made to allow for the pending jump. Finally, an adjustment is made in the levels of the break points between the different conditions when time is saved through rote learning (see below) and when the total number of pieces on the board falls below an arbitrary number. All break points are determined by single data words which can be changed at any time by manual intervention.

This tying of the ply with board conditions achieves three desired results. In the first place, it permits board evaluations to be made under conditions of relative stability for so-called dead positions, as defined by Turing.[15] Secondly, it causes greater surveillance of those paths which offer better opportunities for gaining or losing an advantage. Finally, since branching is usually seriously restricted by a jump situation, the total number of board positions and moves to be considered is still held down to a reasonable number and is more equitably distributed between the various possible initial moves.

As a practical matter, machine-playing time usually has been limited to approximately 30 seconds per move. Elaborate table-lookup procedures, fast sorting and searching procedures, and a variety of new programming tricks were developed, and full use was made of all of the resources of the IBM 704 to increase the operating speed as much as possible. One can, of course, set the playing time at any desired value by adjustments of the permitted ply; too small a ply results in a bad game and too large a ply makes the game unduly costly in terms of machine time.

- *Other modes of play*

For study purposes the program was written to accommodate several variations of this basic plan. One of these permits the program to play against itself, that is, to play both sides of the game. This mode of play has been found to be especially good during the early stages of learning.

The program can also follow book games presented to it either on cards or on magnetic tape. When operating in this mode, the program decides at each point in the game on its next move in the usual way and reports this proposed move. Instead of actually making this move, the program refers to the stored record of a book game and makes the book move. The program records its evaluation of the two moves, and it also counts and reports the number of possible moves which the program

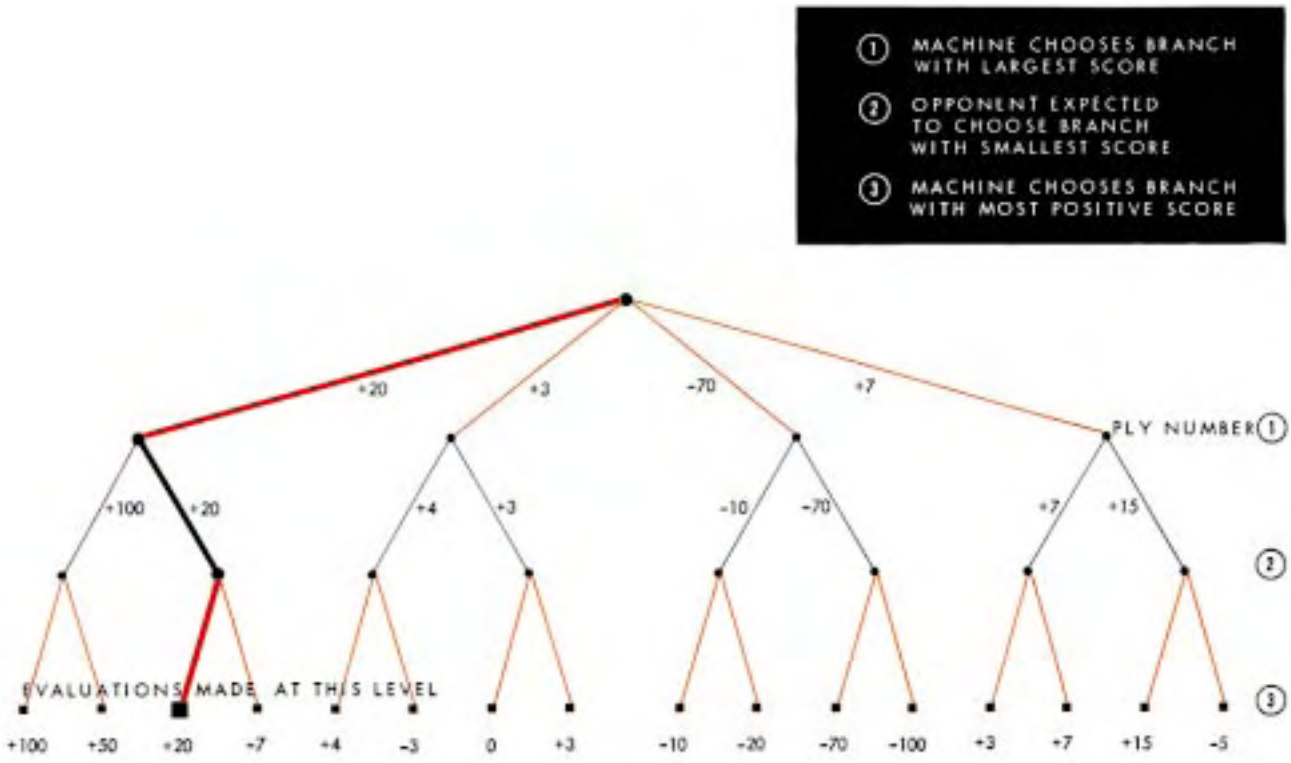


Figure 2 **Simplified diagram showing how the evaluations are backed-up through the "tree" of possible moves to arrive at the best next move. The evaluation process starts at ③.**

rates as being better than the book move and the number it rates as being poorer. The sides are then reversed and the process is repeated. At the end of a book game a correlation coefficient is computed, relating the machine's indicated moves to those moves adjudged best by the checker masters.[16]

It should be noted that the emphasis throughout all of these studies has been on learning techniques. The temptation to improve the machine's game by giving it standard openings or other man-generated knowledge of playing techniques has been consistently resisted. Even when book games are played, no weight is given to the fact that the moves as listed are presumably the best possible moves under the circumstances.

For demonstration purposes, and also as a means of avoiding lost machine time while an opponent is thinking, it is sometimes convenient to play several simultaneous games against different opponents. With the program in its present form the most convenient number for this purpose has been found to be six, although eight have been played on a number of occasions.

Games may be started with any initial configuration for the board position so that the program may be tested on end games, checker puzzles, et cetera. For nonstandard starting conditions, the program lists the initial piece arrangement. From time to time, and at the end of each game, the program also tabulates various bits of statistical information which assist in the evaluation of playing performance.

Numerous other features have also been added to make the program convenient to operate (for details see Appendix A), but these have no direct bearing on the problem of learning, to which we will now turn our attention.

Rote learning and its variants

Perhaps the most elementary type of learning worth discussing would be a form of rote learning in which the program simply saved all of the board positions encountered during play, together with their computed scores. Reference could then be made to this memory record and a certain amount of computing time might be saved. This can hardly be called a very advanced form of learning; nevertheless, if the program then utilizes the saved time to compute further in depth it will improve with time.

Fortunately, the ability to store board information at a ply of 0 and to look up boards at a larger ply provides the possibility of looking much farther in advance than might otherwise be possible. To understand this, consider a very simple case where the look-ahead is always terminated at a fixed ply, say 3. Assume further that the program saves only the board positions encountered during the actual play with their associated backed-up

scores. Now it is this list of previous board positions that is used to look up board positions while at a ply level of 3 in the subsequent games. If a board position is found, its score has, in effect, already been backed up by three levels, and if it becomes effective in determining the move to be made, it is a 6-ply score rather than a simple 3-ply score. This new initial board position with its 6-ply score is, in turn, saved and it may be encountered in a future game and the score backed up by an additional set of three levels, et cetera. This procedure is illustrated in Fig. 3. The incorporation of this variation, together with the simpler rote-learning feature, results in a fairly powerful learning technique which has been studied in some detail.

Several additional features had to be incorporated into the program before it was practical to embark on learning studies using this storage scheme. In the first place, it was necessary to impart a sense of direction to the program in order to force it to press on toward a win. To illustrate this, consider the situation of two kings against one king, which is a winning combination for practically all variations in board positions. In time, the program can be assumed to have stored all of these variations, each associated with a winning score. Now, if such a situation is encountered, the program will look ahead along all possible paths and each path will lead to a winning combination, in spite of the fact that only one of the possible initial moves may be along the direct path toward the win while all of the rest may be wasting time. How is the program to differentiate between these?

A good solution is to keep a record of the ply value of the different board positions at all times and to make a further choice between board positions on this basis. If ahead, the program can be arranged to push directly toward the win while, if behind, it can be arranged to adopt delaying tactics. The most recent method used is to carry the effective ply along with the score by simply decreasing the magnitude of the score a small amount each time it is backed-up a ply level during the analyses. If the program is now faced with a choice of board positions whose scores differ only by the ply number, it will automatically make the most advantageous choice, choosing a low-ply alternative if winning and a high-ply alternative if losing. The significance of this concept of a direction sense should not be overlooked. Even without "learning," it is very important. Several of the early attempts at learning failed because the direction sense was not properly taken into account.

- *Cataloging and culling stored information*

Since practical considerations limit the number of board positions which can be saved, and since the time to search through those that are saved can easily become unduly long, one must devise systems (1) to catalog boards that are saved, (2) to delete redundancies, and (3) to discard board positions which are not believed to be of much value. The most effective cataloging system found to date starts by standardizing all board positions, first by reversing the pieces and piece positions if it is a board position in which White is to move, so that all boards are reported as if it were Black's turn to move. This reduces by nearly a factor of two the number of boards which must be saved. Board positions, in which all of the pieces are kings, can be reflected about the diagonals with a possible fourfold reduction in the number which must be saved. A more compact board representation than the one employed during play is also used so as to minimize the storage requirements.

After the board positions are standardized, they are grouped into records on the basis of (1) the number of pieces on the board, (2) the presence or absence of a piece advantage, (3) the side possessing this advantage, (4) the presence or absence of kings on the board, (5) the side having the so-called "move," or opposition advantage, and finally (6) the first moments of the pieces about normal and diagonal axes through the board. During play, newly acquired board positions are saved in the memory until a reasonable number have been accumulated, and they are then merged with those on the "memory tape" and a new memory tape is produced. Board positions within a record are listed in a serial fashion, being sorted with respect to the words which define them. The records are arranged on the tape in the order that they are most likely to be needed during the course of a game; board positions with 12 pieces to a side coming first, et cetera. This method of cataloging is very important because it cuts tape-searching time to a minimum.

Reference must be made, of course, to the board positions already saved, and this is done by reading the correct record into the memory and searching through it by a dichotomous search procedure. Usually five or more records are held in memory at one time, the exact number at any time depending upon the lengths of the particular records in question. Normally, the program calls three or four new records into memory during each new move, making room for them as needed, by discarding the records which have been held the longest.

Two different procedures have been found to be of value in limiting the number of board positions that are saved; one based on the frequency of use, and the second on the ply. To keep track of the frequency of use, an age term is carried along with the score. Each new board position to be saved is arbitrarily assigned an age. When reference is made to a stored board position, either to update its score or to utilize it in the look-ahead procedure, the age recorded for this board position is divided by two. This is called *refreshing*. Offsetting this, each board position is automatically aged by one unit at the memory merge times (normally occurring about once every 20 moves). When the age of any one board position reaches an arbitrary maximum value this board position is expunged from the record. This is a form of *forgetting*. New board positions which remain unused are soon forgotten, while board positions which are used several times in succession will be refreshed to such an extent that they will be remembered even if not used thereafter for a fairly long period of time. This form of refreshing and forgetting was adopted on the basis of

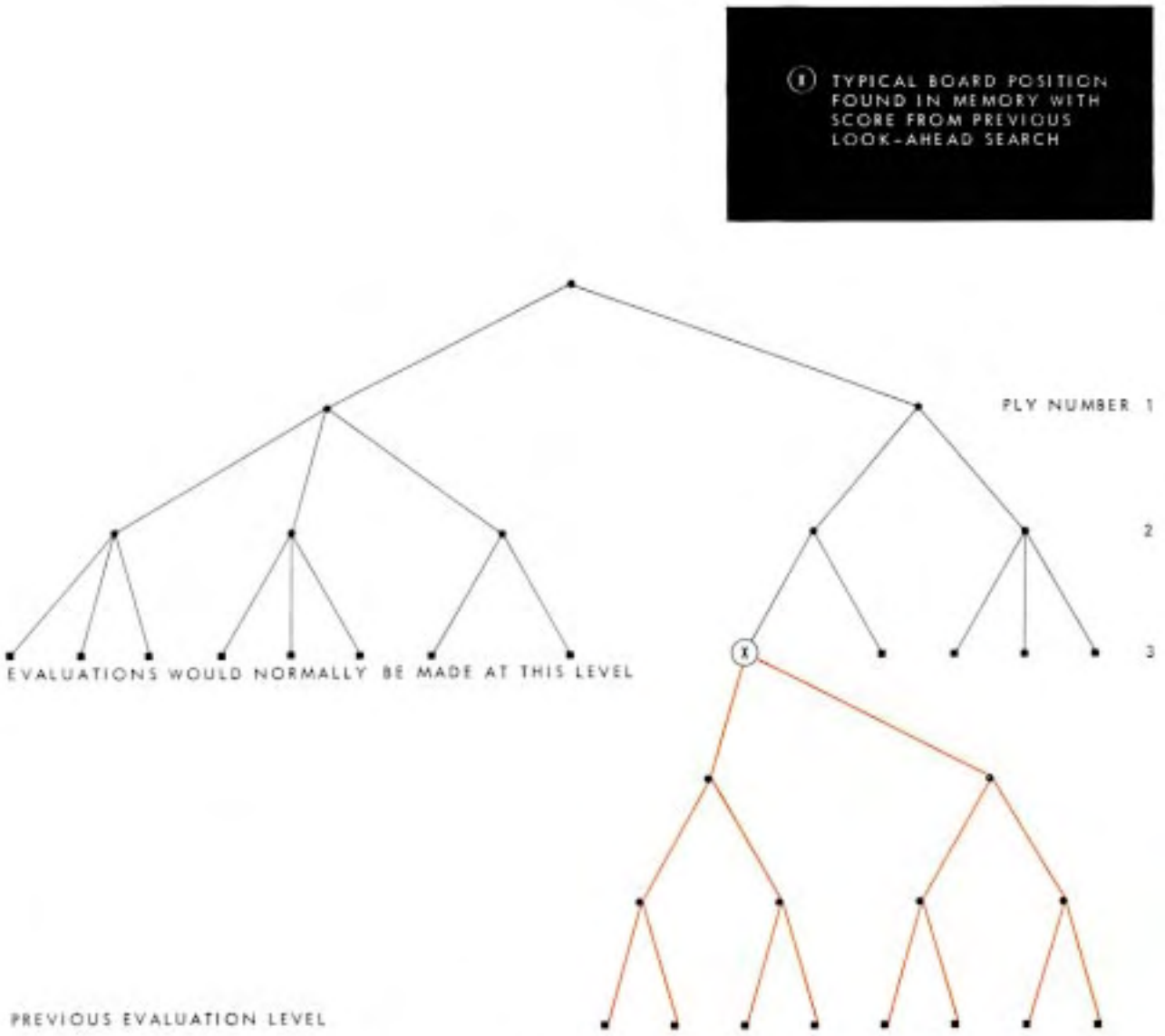


Figure 3 **Simplified representation of the rote-learning process, in which information saved from a previous game is used to increase the effective ply of the backed-up score.**

reflections as to the frailty of human memories. It has proven to be very effective.

In addition to the limitations imposed by forgetting, it seemed desirable to place a restriction on the maximum size of any one record. Whenever an arbitrary limit is reached, enough of the lowest-ply board positions are automatically culled from the record to bring the size well below the maximum.

Before embarking on a study of the learning capabilities of the system as just described, it was, of course, first necessary to fix the terms and coefficients in the evaluation polynomial. To do this, a number of different sets of values were tested by playing through a series of book games and computing the move correlation coefficients. These values varied from 0.2 for the poorest polynomial tested, to approximately 0.6 for the one finally adopted. The selected polynomial contained four terms (as contrasted with the use of 16 terms in later experiments). In decreasing order of importance these were: (1) piece advantage, (2) denial of occupancy, (3) mobility, and (4) a hybrid term which combined control of the center and piece advancement.

- *Rote-learning tests*

After a scoring polynomial was arbitrarily picked, a series of games was played, both self-play and play against many different individuals (several of these being checker masters). Many book games were also followed, some

of these being end games. The program learned to play a very good opening game and to recognize most winning and losing end positions many moves in advance, although its midgame play was not greatly improved. This program now qualifies as a rather better-than-average novice, but definitely not as an expert.

At the present time the memory tape contains something over 53,000 board positions (averaging 3.8 words each) which have been selected from a much larger number of positions by means of the culling techniques described. While this is still far from the number which would tax the listing and searching procedures used in the program, rough estimates, based on the frequency with which the saved boards are utilized during normal play (these figures being tabulated automatically), indicate that a library tape containing at least 20 times the present number of board positions would be needed to improve the midgame play significantly. At the present rate of acquisition of new positions this would require an inordinate amount of play and, consequently, of machine time.[17]

The general conclusions which can be drawn from these tests are that:

(1) An effective rote-learning technique must include a procedure to give the program a sense of direction, and it must contain a refined system for cataloging and storing information.

(2) Rote-learning procedures can be used effectively on machines with the data-handling capacity of the IBM 704 if the information which must be saved and searched does not occupy more than, roughly, one million words, and if not more than one hundred or so references need to be made to this information per minute. These figures are, of course, highly dependent upon the exact efficiency of cataloging which can be achieved.

(3) The game of checkers, when played with a simple scoring scheme and with rote learning only, requires more than this number of words for master caliber of play and, as a consequence, is not completely amenable to this treatment on the IBM 704.

(4) A game, such as checkers, is a suitable vehicle for use during the development of learning techniques, and it is a very satisfactory device for demonstrating machine-learning procedures to the unbelieving.

Learning procedure involving generalizations

An obvious way to decrease the amount of storage needed to utilize past experience is to generalize on the basis of experience and to save only the generalizations. This should, of course, be a continuous process if it is to be truly effective, and it should involve several levels of abstraction. A start has been made in this direction by having the program select a subset of possible terms for use in the evaluation polynomial and by having the program determine the sign and magnitude of the coefficients which multiply these parameters. At the present time this subset consists of 16 terms chosen from a list of 38 parameters. The piece-advantage term needed to define the task is computed separately and, of course, is not altered by the program.

After a number of relatively unsuccessful attempts to have the program generalize while playing both sides of the game, the program was arranged to act as two different players, for convenience called *Alpha* and *Beta*. Alpha generalizes on its experience after each move by adjusting the coefficients in its evaluation polynomial and by replacing terms which appear to be unimportant by new parameters drawn from a reserve list. Beta, on the contrary, uses the same evaluation polynomial for the duration of any one game. Program Alpha is used to play against human opponents, and during self-play Alpha and Beta play each other.

At the end of each self-play game a determination is made of the relative playing ability of Alpha, as compared with Beta, by a neutral portion of the program. If Alpha wins—or is adjudged to be ahead when a game is otherwise terminated—the then current scoring system used by Alpha is given to Beta. If, on the other hand, Beta wins or is ahead, this fact is recorded as a black mark for Alpha. Whenever Alpha receives an arbitrary number of black marks (usually set at three) it is assumed to be on the wrong track, and a fairly drastic and arbitrary change is made in its scoring polynomial (by reducing the coefficient of the leading term to zero). This action is necessary on occasion, since the entire learning process is an attempt to find the highest point in multidimensional scoring space in the presence of many secondary maxima on which the program can become trapped. By manual intervention it is possible to return to some previous condition or make some other change if it becomes apparent that the learning process is not functioning properly. In general, however, the program seeks to extricate itself from traps and to improve more or less continuously.

The capability of the program can be tested at any time by having Alpha play one or more book games (with the learning procedure temporarily immobilized) and by correlating its play with the recommendations of the masters or, more interestingly, by pitting it against a human player.

- *Polynomial modification procedure*

If Alpha is to make changes in its scoring polynomial, it must be given some trustworthy criteria for measuring performance. A logical difficulty presents itself, since the only measuring parameter available is this same scoring polynomial that the process is designed to improve. Recourse is had to the peculiar property of the look-ahead procedure, which makes it less important for the scoring polynomial to be particularly good the further ahead the process is continued. This means that one can evaluate the relative change in the positions of two players, when this evaluation is made over a fairly large number of moves, by using a scoring system which is much too gross to be significant on a move-by-move basis.

Perhaps an even better way of looking at the matter

is that we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably will occur during actual play. Of course, if one could develop a perfect system of this sort it would be the equivalent of always looking ahead to the end of the game. The nearer this ideal is approached, the better would be the play.[18]

In order to obtain a sufficiently large span to make use of this characteristic, Alpha keeps a record of the apparent goodness of its board positions as the game progresses. This record is kept by computing the scoring polynomial for each board position encountered in actual play and by saving this polynomial in its entirety. At the same time, Alpha also computes the backed-up score for all board positions, using the look-ahead procedure described earlier. At each play by Alpha the initial board score, as saved from the previous Alpha move, is compared with the backed-up score for the current position. The difference between these scores, defined as *delta*, is used to check the scoring polynomial. If delta is positive it is reasonable to assume that the initial board evaluation was in error and terms which contributed positively should have been given more weight, while those that contributed negatively should have been given less weight. A converse statement can be made for the case where delta is negative. Presumably, in this case, either the initial board evaluation was incorrect, or a wrong choice of moves was made, and greater weight should have been given to terms making negative contributions, with less weight to positive terms. These changes are not made directly but are brought about in an involved way which will now be described.

A record is kept of the correlation existing between the signs of the individual term contributions in the initial scoring polynomial and the sign of delta. After each play an adjustment is made in the values of the correlation coefficients, due account being taken of the number of times that each particular term has been used and has had a nonzero value. The coefficient for the polynomial term (other than the piece-advantage term) with the then largest correlation coefficient is set at a prescribed maximum value with proportionate values determined for all of the remaining coefficients. Actually, the term coefficients are fixed at integral powers of 2, this power being defined by the ratio of the correlation coefficients. More precisely, if the ratio of two correlation coefficients is equal to or larger than n but less than $n+1$, where n is an integer, then the ratio of the two term coefficients is set equal to 2^n. This procedure was adopted in order to increase the range in values of the term coefficients. Whenever a correlation-coefficient calculation leads to a negative sign, a corresponding reversal is made in the sign associated with the term itself.

- *Instabilities*

It should be noted that the span of moves over which delta is computed consists of a remembered part and an anticipated portion. During the remembered play, use had been made of Alpha's current scoring polynomial to determine Alpha's moves but not to determine the opponent's moves, while during the anticipation play the moves for both sides are made using Alpha's scoring polynomial. One is tempted to increase the sensitivity of delta as an indicator of change by increasing the span of the remembered portion. This has been found to be dangerous since the coefficients in the evaluation polynomial and, indeed, the terms themselves, may change between the time of the remembered evaluation and the time at which the anticipation evaluation is made. As a matter of fact, this difficulty is present even for a span of one move-pair. It is necessary to recompute the scoring polynomial for a given initial board position after a move has been determined and after the indicated corrections in the scoring polynomial have been made, and to save this score for future comparisons, rather than to save the score used to determine the move. This may seem a trivial point, but its neglect in the initial stages of these experiments led to oscillations quite analogous to the instability induced in electrical circuits by long delays in a feedback loop.

As a means of stabilizing against minor variations in the delta values, an arbitrary minimum value was set, and when delta fell below this minimum for any particular move no change was made in the polynomial. This same minimum value is used to set limits for the initial board evaluation score to decide whether or not it will be assumed to be zero. This minimum is recomputed each time and, normally, has been fixed at the average value of the coefficients for the terms in the currently existing evaluation polynomial.

Still another type of instability can occur whenever a new term is introduced into the scoring polynomial. Obviously, after only a single move the correlation coefficient of this new term will have a magnitude of 1, even though it might go to 0 after the very next move. To prevent violent fluctuations due to this cause, the correlation coefficients for newly introduced terms are computed as if these terms had already been used several times and had been found to have a zero correlation coefficient. This is done by replacing the times-used number in the calculation by an arbitrary number (usually set at 16) until the usage does, in fact, equal this number.

After a term has been in use for some time, quite the opposite action is desired so that the more recent experience can outweigh earlier results. This is achieved, together with a substantial reduction in calculation time, by using powers of 2 in place of the actual times-used and by limiting the maximum power that is used. To be specific, at any stage of play defined as the N^{th} move, corrections to the values of the correlation coefficients C_N are made using 16 for N until N equals 32, whereupon 32 is used until N equals 64, et cetera, using the formula:

$$C_N = C_{N-1} - \frac{C_{N-1} \pm 1}{N},$$

and a value for N larger than 256 is never used.

After a minimum was set for delta it seemed reasona-

ble to attach greater weight to situations leading to large values of delta. Accordingly, two additional categories are defined. If a contribution to delta is made by the first term, meaning that a change has occurred in the piece ratio, the indicated changes in the correlation coefficients are doubled, while if the value of delta is so large as to indicate that an almost sure win or lose will result, the effect on the correlation coefficients is quadrupled.

- *Term replacement*

Mention has been made several times of the procedure for replacing terms in the scoring polynomial. The program, as it is currently running, contains 38 different terms (in addition to the piece-advantage term), 16 of these being included in the scoring polynomial at any one time and the remaining 22 being kept in reserve. After each move a low-term tally is recorded against that active term which has the lowest correlation coefficient and, at the same time, a test is made to see if this brings its tally count up to some arbitrary limit, usually set at 8. When this limit is reached for any specific term, this term is transferred to the bottom of the reserve list, and it is replaced by a term from the head of the reserve list. This new term enters the polynomial with zero values for its correlation coefficient, times used, and low-tally count. On the average, then, an active term is replaced once each eight moves and the replaced terms are given another chance after 176 moves. As a check on the effectiveness of this procedure, the program reports on the usage which has accrued against each discarded term. Terms which are repeatedly rejected after a minimum amount of usage can be removed and replaced with completely new terms.

It might be argued that this procedure of having the program select terms for the evaluation polynomial from a supplied list is much too simple and that the program should generate the terms for itself. Unfortunately, no satisfactory scheme for doing this has yet been devised. With a man-generated list one might at least ask that the terms be members of an orthogonal set, assuming that this has some meaning as applied to the evaluation of a checker position. Apparently, no one knows enough about checkers to define such a set. The only practical solution seems to be that of including a relatively large number of possible terms in the hope that all of the contributing parameters get covered somehow, even though in an involved and redundant way. This is not an undesirable state of affairs, however, since it simulates the situation which is likely to exist when an attempt is made to apply similar learning techniques to real-life situations.

Many of the terms in the existing list are related in some vague way to the parameters used by checker experts. Some of the concepts which checker experts appear to use have eluded the writer's attempts at definition, and he has been unable to program them. Some of the terms are quite unrelated to the usual checker lore and have been discovered more or less by accident. The second moment about the diagonal axis through the double corners is an example. Twenty-seven different simple terms are now in use, the rest being combinational terms, as will be described later.

A word might be said about these terms with respect to the exact way in which they are defined and the general procedures used for their evaluation. Each term relates to the relative standings of the two sides, with respect to the parameter in question, and it is numerically equal to the difference between the ratings for the individual sides. A reversal of the sign obviously corresponds to a change of sides. As a further means of insuring symmetry the individual ratings of the respective sides are determined at corresponding times in the play as viewed by the side in question. For example, consider a parameter which relates to the board conditions as left after one side has moved. The rating of Black for such a parameter would be made after Black had moved, and the rating of White would not be made until after White had moved. During anticipation play, these individual ratings are made after each move and saved for future reference. When an evaluation is desired the program takes the differences between the most recent ratings and those made a move earlier. In general, an attempt has been made to define all parameters so that the individual-side ratings are expressible as small positive integers.

- *Binary connective terms*

In addition to the simple terms of the type just described, a number of combinational terms have been introduced. Without these terms the scoring polynomial would, of course, be linear. A number of different ways of introducing nonlinear terms have been devised but only one of these has been tested in any detail. This scheme provides terms which have some of the properties of binary logical connectives. Four such terms are formed for each pair of simple terms which are to be related. This is done by making an arbitrary division of the range in values for each of the simple terms and assigning the binary values of 0 and 1 to these ranges. Since most of the simple terms are symmetrical about 0, this is easily done on a sign basis. The new terms are then of the form $A \cdot B$, $A \cdot \overline{B}$, $\overline{A} \cdot B$, and $\overline{A} \cdot \overline{B}$, yielding values either of 0 or 1. These terms are introduced into the scoring polynomial with adjustable coefficients and signs, and are thereafter indistinguishable from the other terms.

As it would require some 1404 such combinational terms to interrelate the 27 simple terms originally used, it was found desirable to limit the actual number of combinational terms used at any one time to a small fraction of these and to introduce new terms only as it became possible to retire older ineffectual terms. The terms actually used are given in Appendix C.

- *Preliminary learning-by-generalization tests*

An idea of the learning ability of this procedure can be gained by analyzing an initial test series of 28 games[19] played with the program just described. At the start an arbitrary selection of 16 terms was chosen and all terms

were assigned equal weights. During the first 14 games Alpha was assigned the White side, with Beta constrained as to its first move (two cycles of the seven different initial moves). Thereafter, Alpha was assigned Black and White alternately. During this time a total of 29 different terms was discarded and replaced, the majority of these on two different occasions.

Certain other figures obtained during these 28 games are of interest. At frequent intervals the program lists the 12 leading terms in Alpha's scoring polynomial with their correlation coefficients and a running count of the number of times these coefficients have been altered. Based on these samplings, one observes that at least 20 different terms were assigned the largest coefficient at some time or other, some of these alternating with other terms a number of times, and two even reappearing at the top of the list with their signs reversed. While these variations were more violent at the start of the series of games and decreased as time went on, their presence indicated that the learning procedure was still not completely stable. During the first seven games there were at least 14 changes in occupancy at the top of the list involving 10 different terms. Alpha won three of these games and lost four. The quality of the play was extremely poor. During the next seven games there were at least eight changes made in the top listing involving five different terms. Alpha lost the first of these games and won the next six. Quality of play improved steadily but the machine still played rather badly. During Games 15 through 21 there were eight changes in the top listing involving five terms; Alpha winning five games and losing two. Some fairly good amateur players who played the machine during this period agreed that it was "tricky but beatable". During Games 22 through 28 there were at least four changes involving three terms. Alpha won two games and lost five. The program appeared to be approaching a quality of play which caused it to be described as "a better-than-average player". A detailed analysis of these results indicated that the learning procedure did work and that the rate of learning was surprisingly high, but that the learning was quite erratic and none too stable.

- *Second series of tests*

Some of the more obvious reasons for this erratic behavior in the first series of tests have been identified. The program was modified in several respects to improve the situation, and additional tests were made. Four of these modifications are important enough to justify a detailed explanation.

In the first place, the program was frequently fooled by bad play on the part of its opponent. A simple solution was to change the correlation coefficients less drastically when delta was positive than when delta was negative. The procedure finally adopted for the positive delta case was to make corrections to selected terms in the polynomial only. When the scoring polynomial was positive, changes were made to coefficients associated with the negatively contributing terms, and when the polynomial was negative, changes were made to the coefficients associated with positively contributing terms. No changes were made to coefficients associated with terms which happened to be zero. For the negative delta case, changes were made to the coefficients of all contributing terms, just as before.

A second defect seemed to be connected with the too frequent introduction of new terms into the scoring polynomial and the tendency for these new terms to assume dominant positions on the basis of insufficient evidence. This was remedied by the simple expedient of decreasing the rate of introduction of new terms from one every eight moves to one every 32 moves.

The third defect had to do with the complete exclusion from consideration of many of the board positions encountered during play by reason of the minimum limit on delta. This resulted in the misassignment of credit to those board positions which permitted spectacular moves when the credit rightfully belonged to earlier board positions which had permitted the necessary groundlaying moves. Although no precise way has yet been devised to insure the correct assignment of credit, a very simple expedient was found to be most effective in minimizing the adverse effects of earlier assignments. This expedient was to allow the span of remembered moves, over which delta is computed, to increase until delta exceeded the arbitrary minimum value, and then to apply the corrections to the coefficients as dictated by the terms in the retained polynomial for this earlier board position. In this case, the difficulty which was mentioned in the section on Instabilities in connection with an arbitrary increase in span, does not occur after each correction, since no changes are made in the coefficients of the scoring polynomial as long as delta is below the minimum value. Of course, whenever delta does exceed the minimum value the program must then recompute the initial scoring polynomial for the then current board position and so restart the procedure with a span of a single remembered move-pair. This over-all procedure rectifies the defect of assigning credit to a board position that lies too far along the move chain, but it introduces the possibility of assigning credit to a board position that is not far enough along.

As a partial expedient to compensate for this newly introduced danger, a change was made in the initial board evaluation. Instead of evaluating the initial board positions directly, as was done before, a standard but rudimentary tree-search (terminated after the first non-jump move) was used. Errors due to impending jump situations were eliminated by this procedure, and because of the greater accuracy of the evaluation it was possible to reduce the minimum delta limit by a small amount.

Finally, to avoid the danger of having Beta adopt Alpha's polynomial as a result of a chance win on Alpha's part (or perhaps a situation in which Alpha had allowed its polynomial to degenerate after an early or midgame advantage had been gained), it was decided

to require a majority of wins on Alpha's part before Beta would adopt Alpha's scoring polynomial.

With these modifications, a new series of tests was made. In order to reduce the learning time, the initial selection of terms was made on the basis of the results obtained during the earlier tests, but no attention was paid to their previously assigned weights. In contrast with the earlier erratic behavior, the revised program appeared to be extremely stable, perhaps at the expense of a somewhat lower initial learning rate. The way in which the character of the evaluation polynomial altered as learning progressed is shown in Fig. 4.

The most obvious change in behavior was in regard to the relative number of games won by Alpha and the prevalence of draws. During the first 28 games of the earlier series Alpha won 16 and lost 12. The corresponding figures for the first 28 games of the new series were 18 won by Alpha, and four lost, with six draws. In all cases the games were terminated, if not finished, in 70 moves and a judgment made in terms of the final positions. Unfortunately, these figures are not strictly comparable because of the decreased frequency with which Beta adopted Alpha's polynomial during the second series, both by design and because a programming error immobilized the adoption procedure during part of the tests. Nevertheless, the great decrease in the number of losses and the prevalence of draws seemed to indicate that the learning process was much more stable. Some typical games from this second series are given in Appendix B.

As learning proceeds, it should become harder and harder for Alpha to improve its game, and one would expect the number of wins by Alpha to decrease with time. If secondary maxima in scoring space are encountered, one might even find situations in which Alpha wins less than half of the games. With Beta at such a maximum any minor change in Alpha's polynomial would result in a degradation of its play, and several oscillations about the maximum might occur before Alpha landed at a point which would enable it to beat Beta. Some evidence of this trend is discernible in the play, although many more games will have to be played before it can be observed with certainty.

The tentative conclusions which can be drawn from these tests are:

(1) A simple generalization scheme of the type here used can be an effective learning device for problems amenable to tree-searching procedures.

(2) The memory requirements of such schemes are quite modest and remain fixed with time.

(3) The operating times are also reasonable and remain fixed, independent of the amount of accumulated learning.

(4) Incipient forms of instability in the solution can be expected but, at least for the checker program, these can be dealt with by quite straightforward procedures.

(5) Even with the incomplete and redundant set of parameters which have been used to date, it is possible for the computer to learn to play a better-than-average game of checkers in a relatively short period of time.

As a final precautionary note, it should be stated that these experiments have not encompassed a sufficiently large series of games to demonstrate unambiguously that the learning procedure is completely stable or that it will necessarily lead to the best possible choice of parameters and coefficients.

Rote learning vs. generalization

Some interesting comparisons can be made between the playing style developed by the learning-by-generalization program and that developed by the earlier rote-learning procedure. The program with rote learning soon learned to imitate master play during the opening moves. It was always quite poor during the middle game, but it easily learned how to avoid most of the obvious traps during end-game play and could usually drive on toward a win when left with a piece advantage. The program with the generalization procedure has never learned to play in a conventional manner and its openings are apt to be weak. On the other hand, it soon learned to play a good middle game, and with a piece advantage it usually polishes off its opponent in short order. Interestingly enough, after 28 games it had still not learned how to win an end game with two kings against one in a double corner.

Apparently, rote learning is of the greatest help, either under conditions when the results of any specific action are long delayed, or in those situations where highly specialized techniques are required. Contrasting with this, the generalization procedure is most helpful in situations in which the available permutations of conditions are large in number and when the consequences of any specific action are not long delayed.

- *Procedures involving both forms of learning*

The next obvious step is to combine the better features of the rote-learning procedure with a generalization scheme. This must be done with some care, since it is not practical to update the previously saved information after every change in the evaluation polynomial. A compromise solution might be to save only a very limited amount of information during the early stages of learning and to increase the amount as warranted by the increasing stability of the evaluation coefficient with learning. For example, the program could be arranged to save only the piece-advantage term at the start. At some stage in the learning process the next term could be added, perhaps when no change had been made in the parameter used for this term during some fairly long period, say for three complete games. If and when the program is able to play an additional period without changes in the next parameter, this could also be added, et cetera. Whenever a change does occur in a parameter previously assumed to be stable the entire memory tape could be reviewed, all terms involving the changed parameter and those lower on the list could be expunged, and the program could drop back to the earlier condition with respect to its term-saving schedule.

Another solution would be to utilize the generalization scheme alone until it had become fairly stable and to introduce rote learning at this time. It is, of course, perfectly feasible to salvage much of the learning which has been accumulated by both of the programs studied to date. This could be done by appending an abridged form of the present memory tape to the generalization scheme in its present stage of learning and by proceeding from there in accordance with the first solution proposed above.

- *Future development*

While it is believed that these tests have reached the stage of diminishing returns, some effort might well be expended in an attempt to get the program to generate its own parameters for the evaluation polynomial. Lacking a perfectly general procedure, it might still be possible to generate terms based on theories as proposed by students of the game. This procedure would be at variance with the writer's previous philosophy, but it is highly likely that similar compromises will have to be made when one attempts to apply learning procedures to problems of economic importance.

Conclusions

As a result of these experiments one can say with some certainty that it is now possible to devise learning schemes which will greatly outperform an average person and that such learning schemes may eventually be economically feasible as applied to real-life problems.

Acknowledgments

Many different people have contributed to these studies through stimulating discussions of the basic problems. From time to time the writer was assisted by several different programmers, although most of the detailed work was his own. The forbearance of the machine room operators and their willingness to play the machine at all hours of the day and night are also greatly appreciated.

Footnotes and References

1. Some of these are quite profound and have a bearing on the questions raised by Nelson Goodman in *Fact, Fiction and Forecast,* Harvard University Press, 1954.
2. Warren S. McCulloch ("The Brain as a Computing Machine," *Elec. Eng.* **69,** 492, 1949) has compared the digital computer to the nervous system of a flatworm. To extend this comparison to the situation under discussion would be unfair to the worm, since its nervous system is actually quite highly organized as compared with the random-net studies by B. G. Farley and W. A. Clarke ("Simulation of Self-Organizing Systems by Digital Computers," *IRE PGIT* **4,** 76, Sept. 1954), N. Rochester, J. H. Holland, L. H. Haibt and W. L. Duda ("Tests on a Cell Assembly Theory of the Action of the Brain Using a Large Digital Computer," *IRE Transactions on Information Theory,* **IT-2,** No. 3, 80, Sept. 1956), and by F. Rosenblatt ("The Perceptron; A Probabilistic Model for Information Storage and Organization in the Brain," *Psych. Rev.,* **6,** 65, November 1958).
3. The first operating checker program for the IBM 701 was written in 1952. This was recoded for the IBM 704 in 1954. The first program with learning was completed in 1955 and demonstrated on television on February 24, 1956.
4. C. E. Shannon, "Programming a Computer for Playing Chess," *Phil. Mag.* **41,** 256 (March 1950).
5. A. Bernstein and M. deV. Roberts, "Computer vs. Chess-Player," *Scient. Amer.* **198,** 6 (June 1958).
6. J. Kister, P. Stein, S. Ulam, W. Walden, M. Wells, "Experiments in Chess," *Journal of the ACM,* **4,** 174 (April 1957).
7. A. Newell, J. C. Shaw and H. A. Simon, "Chess-Playing Programs and the Problem of Complexity," *IBM J. of Res. & Devel.* **2,** 320 (October 1958).
8. Shannon, *loc cit.*
9. C. S. Strachey, "Logical or Non-Mathematical Programmes," Proc. of ACM Meeting at Toronto, Ontario, pp. 46-49, Sept. 8-10, 1952.
10. One of the more interesting of these was to express a board position in terms of the first and higher moments of the white and black pieces separately about two orthogonal axes on the board. Two such sets of axes were tried, one set being parallel to the sides of the board and the second set being those through the diagonals.
11. This apt phraseology was suggested by John McCarthy.
12. Not the capture of all of the opponent's pieces, as popularly assumed, although nearly all games end in this fashion.
13. The use of a weight ratio rather than this, conforming more closely to the values assumed by many players, can lead into certain logical complications, as found by Strachey, *loc. cit.*
14. The only departure from complete generality of the game as programmed is that the program requires the opponent to make a permissible move, including the taking of a capture if one is offered. "Huffing" is not permitted.
15. B. V. Bowden, *Faster Than Thought,* Chapter 25, Pitman, 1953.
16. This coefficient is defined as $C=(L-H)/(L+H)$, where L is the total number of different legal moves which the machine judged to be poorer than the indicated book moves, and H is the total number which it judged to be better than the book moves.
17. This playing-time requirement, while large in terms of cost, would be less than the time which the checker master probably spends to acquire his proficiency.
18. There is a logical fallacy in this argument. The program might save only invariant terms which have nothing to do with goodness of play; for example, it might count the squares on the checkerboard. The forced inclusion of the piece-advantage term prevents this.
19. Each game averaged 68 moves (34 to a side), of which approximately 20 caused changes to be made in the scoring polynomial.

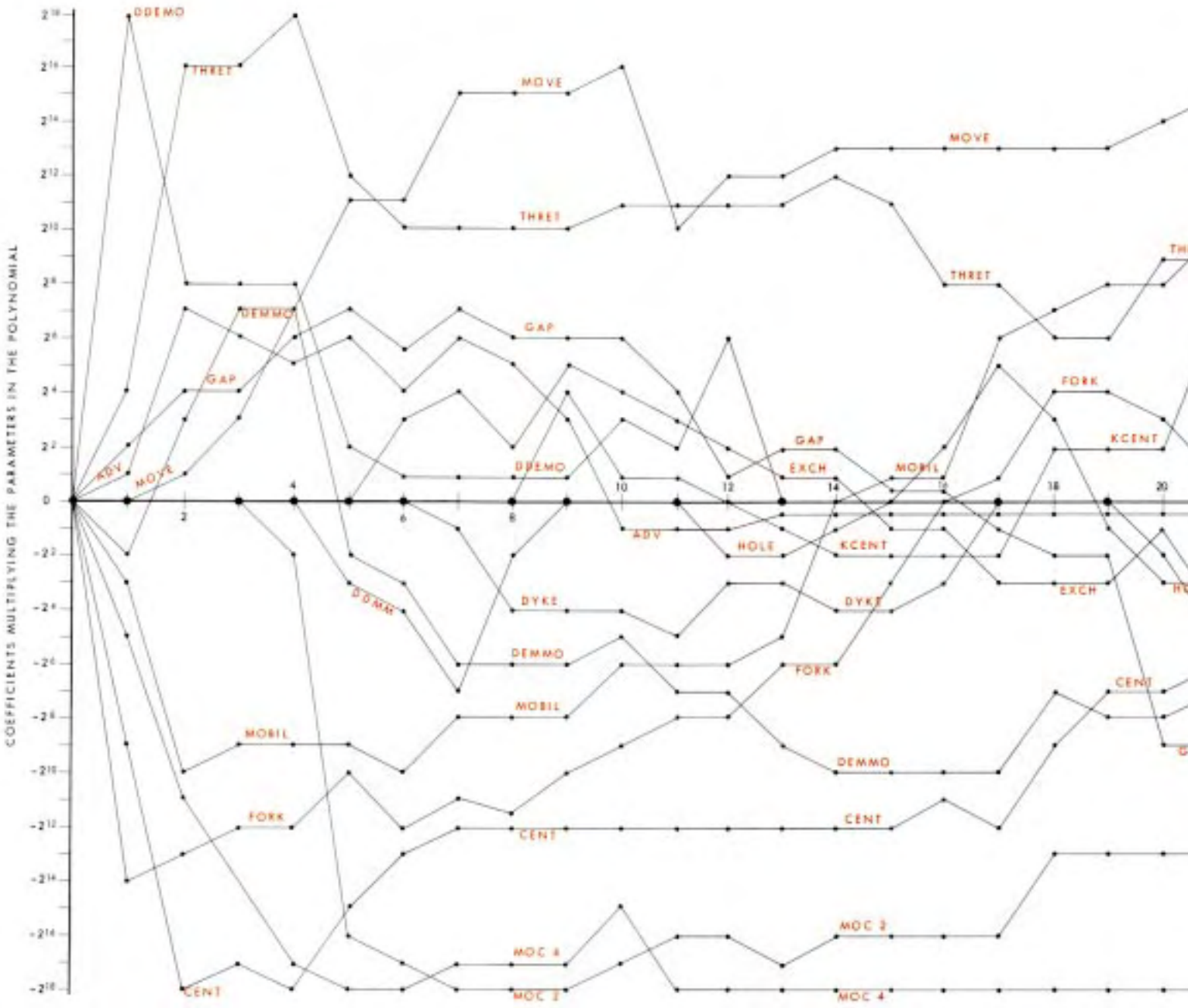
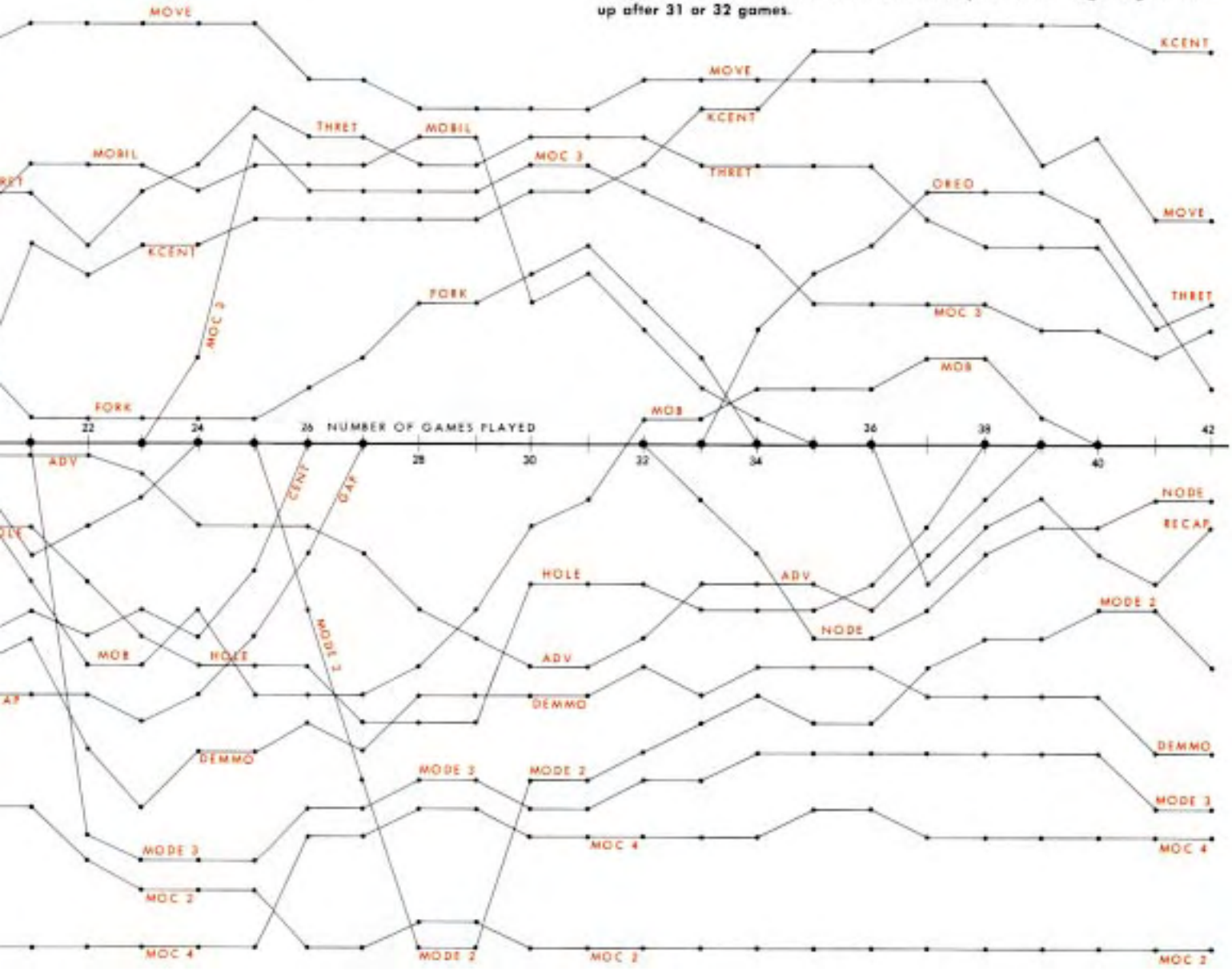

COEFFICIENTS MULTIPLYING THE PARAMETERS IN THE POLYNOMIAL
DDEMO
THREE
MOVE
DEMMO
GAP
ADV
DYKE
MOBIL
FORK
CENT
MOC 3
MOC 4
EXCH
KCENT
HOLE
2
4
6
8
10
12
14
16
18
20

Figure 4 **Second series of learning-by-generalization tests. Coefficients assigned by the program to the more significant parameters of the evaluation polynomial plotted as a function of the number of games played. Two regions of special interest might be noted: (1) the situation after 13 or 14 games, when the program found that the initial signs of many of the terms had been set incorrectly, and (2) the conditions of relative stability which are beginning to show up after 31 or 32 games.**



Appendix A: Programming details

- *Approximate size of program*

Basic checker-playing routine	1100 instructions
Input, move verification and output	1400 instructions
Game starting and terminating routines	600 instructions
Loaders, table generators, dumping, et cetera	850 instructions
Statistical and analytical routines	700 instructions
Rote-learning routines	1500 instructions
Generalization-learning routines	650 instructions
Tables and constants for basic play	700 words
Working space for basic play	2000 words
Working space for generalization learning	500 words
Working space for rote learning	balance of memory

- *Approximate computation times*

To find all available moves from given board position	2.6 milliseconds
To make a single move and find resulting board position	1.5 milliseconds
To evaluate a board position (4 terms)	2.4 milliseconds
To find score for a saved board position (rote learning)	2.3 milliseconds
To evaluate position (with 16 terms for generalization learning)	7.5 milliseconds

- *Board representations*

The standard checkerboard numbering system (see Appendix B) is used in communicating with the machine. A modified numbering system is used for internal computations, the numbers shown on the squares in Fig. A-1 corresponding to the bit positions in an IBM 704 word. Any given board position is represented by four such words; one word (*FA*) containing 1's in those bit positions corresponding to squares containing pieces of the color whose turn it is to move and which normally move in a forward direction. To be specific, if it is Black's turn to move (i.e., if Black is "active") *FA* designates the location of all of Black's pieces, both men and kings. Conversely, if White is active, *FA* designates the location of White's kings only, since White's men can only move in the direction arbitrarily called *backward*. The other words designate, respectively: *BA*, backward active pieces; *FP*, forward passive pieces; and *BP*, backward passive pieces.

To conserve space when writing on tape, three words are used to record board positions with kings, and only two words are used for board positions without kings. These are saved in a standardized form, as explained in the text.

Possible moves are designated by five words; one word to indicate by its sign (with the word itself containing other information) whether the moves are jumps or not. (If a jump is available, only jump moves are saved.) The other four words designate the location of those pieces which can move in the four different diagonal directions: *RF*, for right forward; *LF*, for left forward; *LB*, for left backward; and *RB*, for right backward, respectively.

By reference to Fig. A-1, it will be observed that a right-forward move results in an increase of 4 in the square designation, while a left-forward move results in an increase of 5. Bit positions 9, 18 and 27 do not appear on the board. This notation makes it possible to compute available moves for all pieces simultaneously. Having previously computed a word called *EMPTY*, which contains 1's in locations corresponding to all unoccupied squares, one can compute *RF*, for the normal move case, in four instructions, as listed below (in IBM 704 symbolic language):

CLA	*EMPTY*	(puts word *EMPTY* into the accumulator);
ALS	4	(shifts word to left by 4 positions);
ANA	*FA*	(forms logical AND between *EMPTY* and *FA*);
STO	*RF*	(stores word as newly computed *RF*).

Jump moves are computed by a simple extension of this procedure. Multiple jumps are handled as a sequence of single jumps separated by null-reply moves.

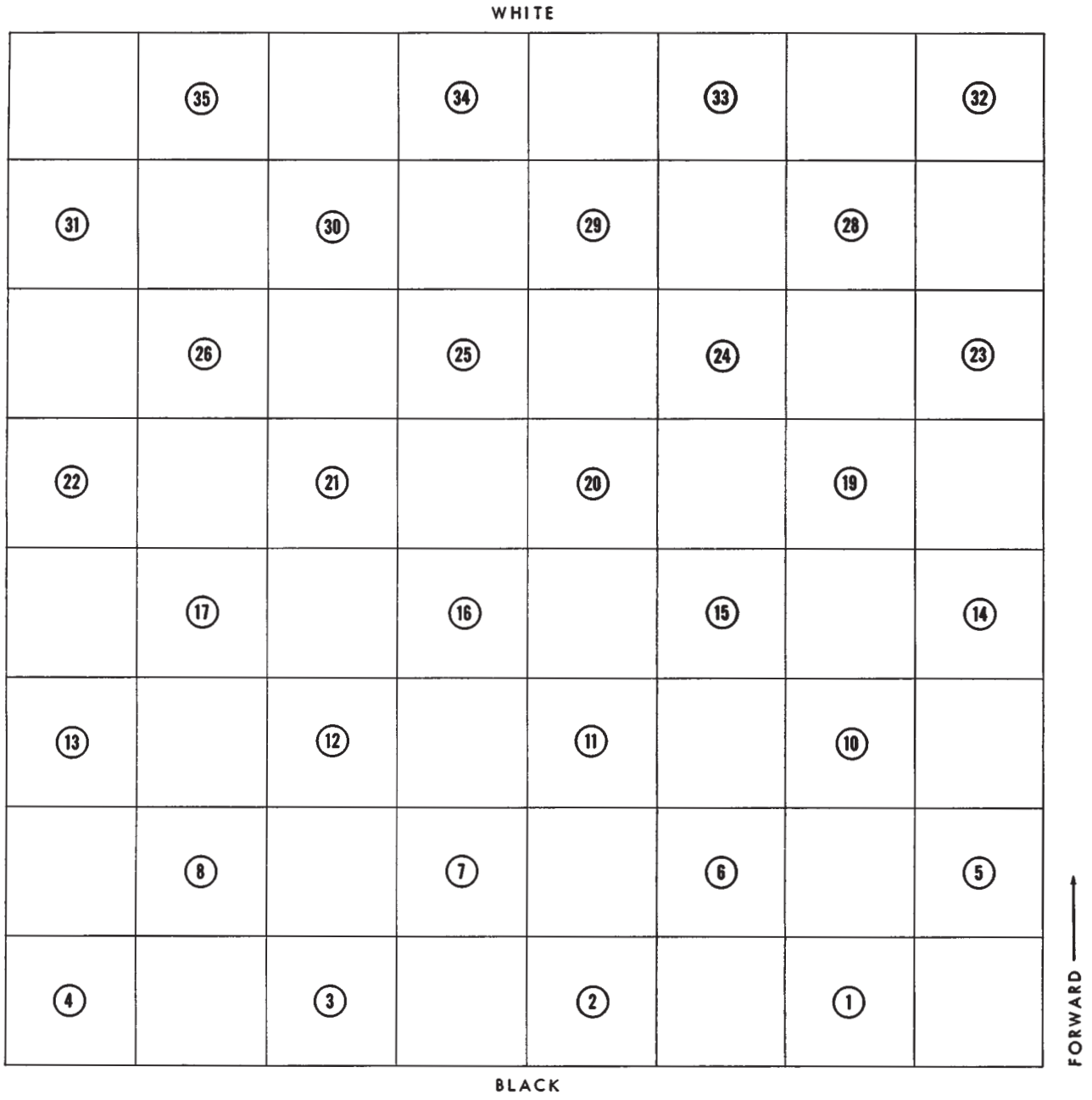


Figure A-1 **Checkerboard notation for internal computations.**

- *Additional time-saving expedients*

Bit counting is done by a table-lookup procedure in a closed subroutine of 16 executed instructions (408 microseconds). This requires a 256-word table which is generated at the start by a 13-word program. Similar table-lookup procedures are used, to turn a word end-for-end, and to locate the 1's in a word for move reporting.

Multiplications are usually avoided. In several places where multiplication by small integers must be done, it is programmed in terms of shifts and logical operations.

During the look-ahead procedure a complete record is kept of the sequence of board positions currently under investigation. As a result, no computing is needed to retract moves.

Appendix B: Sample games from the second series with generalization learning

- *Typical openings*

The first eight moves of selected games in which Alpha played Black against Beta, showing the way in which different types of play were tried.

G-4	*G-6*	*G-12*	*G-17*	*G-19*	*G-21*	*G-31*	*G-37*	*G-39*	*G-41*	*G-43*
10 14	11 16	11 16	11 16	11 16	11 16	11 16	12 16	11 16	10 14	11 16
24 19	22 18	22 17	24 20	24 20	24 20	23 18	24 20	24 20	24 20	23 19
14 18	16 20	16 20	10 14	7 11	8 11	7 11	8 12	10 15	11 15	16 23
23 14	18 14	17 13	20 11	22 17	28 24	27 23	28 24	20 11	27 24	26 19
9 18	9 18	9 14	8 15	10 14	10 14	16 20	10 14	7 16	7 10	8 11
22 15	23 14	23 18	22 17	17 10	23 18	23 19	23 18	21 17	23 18	22 17
11 18	10 17	14 23	7 11	6 15	14 23	20 27	14 23	6 10	14 23	10 14
21 17	21 14	27 18	17 10	28 24	27 18	31 24	27 18	23 19	26 19	17 10

- *Typical games*

Sample games in which Alpha played White against forced Beta openings.

G-1	*G-18*	*G-30*	*G-40*	*G-1*	*G-18*	*G-30*	*G-40*
12 16	12 16	12 16	10 14	9 13	12 16	9 14	4 8
24 19	24 20	24 20	24 20	1 6	24 20	18 9	1 6
8 12	8 12	8 12	11 15	13 17	16 19	8 11	10 14
22 18	28 24	28 24	27 24	32 27	29 25	15 8	6 10
10 14	10 15	10 14	7 10	16 20	13 17	4 11	14 17
26 22	22 18	22 18	23 18	18 14	10 7	19 15	10 15
16 20	15 22	6 10	14 23	11 15	2 11	11 18	17 21
30 26	25 18	24 19	26 19	6 10	14 10	23 14	32 28
11 16	7 10	1 6	10 14	15 18	19 23	13 17	5 9
28 24	18 14	32 28	19 10	14 9	21 14	9 5	27 24
7 11	10 17	3 8	6 15	Terminated	23 26	12 16	20 27
22 17	21 14	26 22	22 17	Manually	10 7	28 24	19 16
3 8	9 18	9 13	2 7		26 30	17 22	12 19
17 10	23 14	18 9	17 10		25 21	6 10	15 22 31
6 15 22	6 9	5 14	7 14		30 26	30 25	9 14
26 17	30 25	22 18	24 19		7 3	1 6	31 26
9 13	9 18	6 9	15 24		11 15	25 21	14 18
17 14	26 23	25 22	28 19		14 10	5 1	28 24
2 7	3 8	2 6	14 17		5 9	21 17	8 11
23 18	23 14	30 25	21 14		10 6	24 20	24 19
16 23	1 6	14 17	9 18		15 19	16 19	21 25
14 10	27 23	21 14 5	25 22		6 1	20 16	30 21
7 14	6 9	6 9	18 25		26 22	17 13	Beta Concedes
18 9	14 10	18 15	29 22		1 6	6 2	
5 14	9 13	11 18	5 9		9 13	13 17	
27 18 9	25 21	20 11 2	31 27		20 16	10 6	
20 27	11 15	10 14	1 5		19 23	Beta Concedes	
31 24	20 11	22 15	20 16		6 9		
12 16	15 18	14 17	3 7		23 27		
21 17	23 14	5 1	22 17		16 11		
13 22	8 15	17 21	8 11		22 25		
25 18	24 19	25 22	17 13		11 7		
1 5	15 24	21 25	11 20		25 30		
9 6	32 28	22 18	13 6		7 2		
5 9	24 27	25 30	7 10		27 32		
6 1	31 24	2 6	6 1		70 Move Termination		



Figure B-1 **Square designations used in reporting games.**

Appendix C: Evaluation polynomial details for second series

- *Method of computing terms*

The 16 terms called for in the evaluation polynomial are computed, individually, by taking the value of the appropriate parameter, as defined below, for the board position under consideration and subtracting the value of this same parameter computed for the board position just prior to the last move (with the necessary reversal in the definitions of active and passive sides). This difference is then multiplied by the corresponding program-computed coefficient, which can vary between -2^{18} and $+2^{18}$, and credited to the side which was passive on the board position under consideration.

• *Definitions of parameters*

ADV (Advancement)
The parameter is credited with 1 for each passive man in the 5th and 6th rows (counting in passive's direction) and debited with 1 for each passive man in the 3rd and 4th rows.

APEX (Apex)
The parameter is debited with 1 if there are no kings on the board, if either square 7 or 26 is occupied by an active man, and if neither of these squares is occupied by a passive man.

BACK (Back Row Bridge)
The parameter is credited with 1 if there are no active kings on the board and if the two bridge squares (1 and 3, or 30 and 32) in the back row are occupied by passive pieces.

CENT (Center Control I)
The parameter is credited with 1 for each of the following squares: 11, 12, 15, 16, 20, 21, 24 and 25 which is occupied by a passive man.

CNTR (Center Control II)
The parameter is credited with 1 for each of the following squares: 11, 12, 15, 16, 20, 21, 24 and 25 that is either currently occupied by an active piece or to which an active piece can move.

CORN (Double-Corner Credit)
The parameter is credited with 1 if the material credit value for the active side is 6 or less, if the passive side is ahead in material credit, and if the active side can move into one of the double-corner squares.

CRAMP (Cramp)
The parameter is credited with 2 if the passive side occupies the cramping square (13 for Black, and 20 for White) and at least one other nearby square (9 or 14 for Black, and 19 or 20 for White), while certain squares (17, 21, 22 and 25 for Black, and 8, 11, 12 and 16 for White) are all occupied by the active side.

DENY (Denial of Occupancy)
The parameter is credited with 1 for each square defined in MOB if on the next move a piece occupying this square could be captured without an exchange.

DIA (Double Diagonal File)
The parameter is credited with 1 for each passive piece located in the diagonal files terminating in the double-corner squares.

DIAV (Diagonal Moment Value)
The parameter is credited with 1/2 for each passive piece located on squares 2 removed from the double-corner diagonal files, with 1 for each passive piece located on squares 1 removed from the double-corner files and with 3/2 for each passive piece in the double-corner files.

DYKE (Dyke)
The parameter is credited with 1 for each string of passive pieces that occupy three adjacent diagonal squares.

EXCH (Exchange)
The parameter is credited with 1 for each square to which the active side may advance a piece and, in so doing, force an exchange.

EXPOS (Exposure)
The parameter is credited with 1 for each passive piece that is flanked along one or the other diagonal by two empty squares.

FORK (Threat of Fork)
The parameter is credited with 1 for each situation in which passive pieces occupy two adjacent squares in one row and in which there are three empty squares so disposed that the active side could, by occupying one of them, threaten a sure capture of one or the other of the two pieces.

GAP (Gap)
The parameter is credited with 1 for each single empty square that separates two passive pieces along a diagonal, or that separates a passive piece from the edge of the board.

GUARD (Back Row Control)
The parameter is credited with 1 if there are no active kings and if either the Bridge or the Triangle of Oreo is occupied by passive pieces.

HOLE (Hole)
The parameter is credited with 1 for each empty square that is surrounded by three or more passive pieces.

KCENT (King Center Control)
The parameter is credited with 1 for each of the following squares: 11, 12, 15, 16, 20, 21, 24 and 25 which is occupied by a passive king.

MOB (Total Mobility)
The parameter is credited with 1 for each square to which the active side could move one or more pieces in the normal fashion, disregarding the fact that jump moves may or may not be available.

MOBIL (Undenied Mobility)
The parameter is credited with the difference between MOB and DENY.

MOVE (Move)
The parameter is credited with 1 if pieces are even with a total piece count (2 for men, and 3 for kings) of less than 24, and if an odd number of pieces are in the move system, defined as those vertical files starting with squares 1, 2, 3 and 4.

NODE (Node)
The parameter is credited with 1 for each passive piece that is surrounded by at least three empty squares.

OREO (Triangle of Oreo)

The parameter is credited with 1 if there are no passive kings and if the Triangle of Oreo (squares 2, 3 and 7 for Black, and squares 26, 30 and 31 for White) is occupied by passive pieces.

POLE (Pole)

The parameter is credited with 1 for each passive man that is completely surrounded by empty squares.

RECAP (Recapture)

This parameter is identical with Exchange, as defined above. (It was introduced to test the effects produced by the random times at which parameters are introduced and deleted from the evaluation polynomial.)

THRET (Threat)

The parameter is credited with 1 for each square to which an active piece may be moved and in so doing threaten the capture of a passive piece on a subsequent move.

- *Binary connective terms*

The abbreviations used for the terms of this type which have been employed are listed below, in the order of $A \cdot B$, $A \cdot \bar{B}$ $\bar{A} \cdot B$, and $\bar{A} \cdot \bar{B}$, where A and B are the two respective parameters heading the sublists of abbreviations.

Denial of Occupancy—Total Mobility	*Undenied Mobility—Denial of Occupancy*	*Undenied Mobility—Center Control I*
DEMO	MODE 1	MOC 1
DEMMO	MODE 2	MOC 3
DDEMO	MODE 3	MOC 2
DDMM	MODE 4	MOC 4

- *Evaluation polynomial (first 12 terms only) after 42 games, during which a total of 1039 different sets of adjustments were made to the terms and their coefficients.**

Term	*Correlation Coefficient*	*Sign of Coefficient*	*Power of 2 Used as Coefficient*	*Times Adjusted*
MOC 2	0.45	−	18	84
KCENT	0.40	+	16	127
MOC 4	0.35	−	14	95
MODE 3	0.33	−	13	210
DEMMO	0.27	−	11	132
MOVE	0.19	+	8	91
ADV	0.19	−	8	739
MODE 2	0.19	−	8	55
BACK	0.14	−	6	6
CNTR	0.13	+	5	12
THRET	0.13	+	5	442
MOC 3	0.10	+	4	89

- *Discarded terms during 42 games**

Term	*Times Adjusted Before Discard*	*Term*	*Times Adjusted Before Discard*
CORN	0	MODE 1	1
CRAMP	0	CENT	386
GUARD	0	MODE 4	0
EXPOS	162	FORK	400
DDMM	19	MOBIL	707
DYKE	115	POLE	11
MOC 1	1	HOLE	598
EXCH	445	GAP	792
DDEMO	53	MOB	608

**Note added in proof:* An additional 20 games have recently been played. Although some significant changes were noted, the general stabilization of the learning process suggested by Figure 4 has been confirmed. During this play, 412 more adjustments were made to the terms and their coefficients and 12 additions were made to the list of discarded terms.

Received March 3, 1959

Arthur L. Samuel

B.A. in Mathematics, College of Emporia, 1923; B.S.E.E. and M.S. at M.I.T., 1926; Honorary Sc.D. from College of Emporia, 1946. Electrical Engineering Instructor at M.I.T., 1926-28. Member of Technical Staff at Bell Telephone Laboratories, 1928-46, where he was mainly in electron-tube research. Professor of Electrical Engineering at University of Illinois, 1946-49. Joined IBM in 1949 and has been concerned with research problems underlying the design, construction and use of digital computers. Is presently Consultant to the Director of Research. Since 1947 has served as expert consultant to the Department of Defense and is Chairman of the Advisory Group on Electron Tubes. He has received awards from the Office of Scientific Research and Development and a commendation from the U. S. Navy. Holds over 50 U. S. patents and more than 200 foreign patents. Received the IRE Papers Award in 1947. He is well known for his design of gas-filled and high-vacuum electron tubes and his work in electron dynamics, solid-state physics and digital computers. Programming the IBM 704 is a mathematical recreation. He is a Fellow of the IRE and APS, and a member of the AIEE, Sigma Xi, AAAS and ACM.