Allen Newell
J. C. Shaw
H. A. Simon

Chess-Playing Programs and the Problem of Complexity

Abstract: This paper traces the development of digital computer programs that play chess. The work of Shannon, Turing, the Los Alamos group, Bernstein, and the authors is treated in turn. The efforts to program chess provide an indication of current progress in understanding and constructing complex and intelligent mechanisms.

Man can sólve problems without knowing how he solves them. This simple fact sets the conditions for all attempts to rationalize and understand human decision making and problem solving. Let us simply assume that it is good to know how to do mechanically what man can do naturally—both to add to man's knowledge of man, and to add to his kit of tools for controlling and manipulating his environment. We shall try to assess recent progress in understanding and mechanizing man's intellectual attainments by considering a single line of attack—the attempts to construct digital computer programs that play chess.

Chess is the intellectual game *par excellence*. Without a chance device to obscure the contest, it pits two intellects against each other in a situation so complex that neither can hope to understand it completely, but sufficiently amenable to analysis that each can hope to outthink his opponent. The game is sufficiently deep and subtle in its implications to have supported the rise of professional players, and to have allowed a deepening analysis through 200 years of intensive study and play without becoming exhausted or barren. Such characteristics mark chess as a natural arena for attempts at mechanization. If one could devise a successful chess machine, one would seem to have penetrated to the core of human intellectual endeavor.

The history of chess programs is an example of the attempt to conceive and cope with complex mechanisms. Now there might have been a trick — one might have discovered something that was as the wheel to the human leg: a device quite different from humans in its methods, but supremely effective in its way, and perhaps very simple. Such a device might play excellent chess, but would fail to further our understanding of human intellectual processes. Such a prize, of course, would be worthy of discovery in its own right, but there appears to be nothing of this sort in sight.

We return to the original orientation: Humans play chess, and when they do they engage in behavior that seems extremely complex, intricate, and successful. Consider, for example, a scrap of a player's (White's) running comment as he analyzes the position in Fig. 1:

> ". . . Are there any other threats? Black also has a threat of Knight to Bishop 5 threatening the Queen, and also putting more pressure on the King's side because his Queen's Bishop can come over after he moves his Knight at Queen 2; however, that is not the immediate threat. Otherwise, his Pawn at King 4 is threatening my Pawn . . ."

Notice that his analysis is qualitative and functional. He wanders from one feature to another, accumulating various bits of information that will be available from time to time throughout the rest of the analysis. He makes evaluations in terms of pressures and immediacies of threat, and gradually creates order out of the situation.

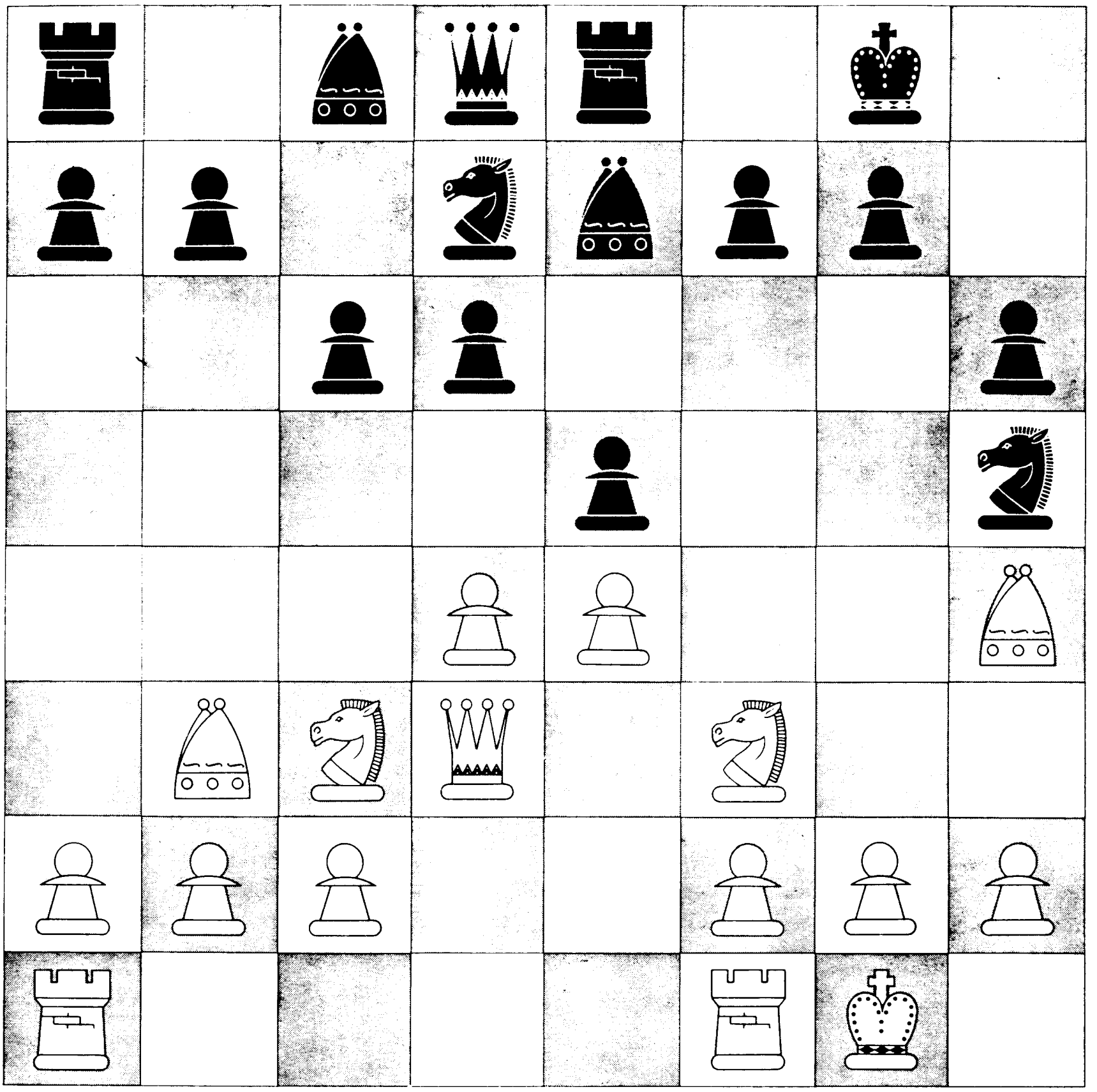
How can we construct mechanisms that will show comparable complexity in their behavior? They need not play in exactly the same way; close simulation of the human is not the immediate issue. But we do assert that complexity of behavior is essential to an intelligent per-

formance—that the complexity of a successful chess program will approach the complexity of the thought processes of a successful human chess player. Complexity of response is dictated by the task, not by idiosyncrasies of the human response mechanism.

There is a close and reciprocal relation between complexity and communication. On the one hand, the complexity of the systems we can specify depends on the language in which we must specify them. Being human, we have only limited capacities for processing information. Given a more powerful language, we can specify greater complexity with limited processing powers.

Let us illustrate this side of the relation between complexity and communication. No one considers building chess machines in the literal sense—fashioning pieces of electronic gear into automatons that will play chess. We think instead of chess programs: specifications written in a language, called machine code, that will instruct a digital computer of standard design how to play chess. There is a reason for choosing this latter course — in addition to any aversion we may have to constructing a large piece of special-purpose machinery. Machine code is a more powerful language than the block diagrams of the electronics engineer. Each symbol in machine code

Figure 1



specifies a larger unit of processing than a symbol in the block diagram. Even a moderately complicated program becomes hopelessly complex if thought of in terms of gates and pulses.

But there is another side to the relation between communication and complexity. We cannot use any old language we please. We must be understood by the person or machine to whom we are communicating. English will not do to specify chess programs because there are no English-understanding computers. A specification in English is a specification to another human who then has the task of creating the machine. Machine code is an advance precisely because there are machines that understand it—because a chess program in machine code is operationally equivalent to a machine that plays chess.

If the machine could understand even more powerful languages, we could use these to write chess programs—and thus get more complex and intelligent programs from our limited human processing capacity. But communication is limited by the intelligence of the least participant, and at present a computer has only passive capability. The language it understands is one of simple commands—it must be told very much about what to do.

Thus it seems that the rise of effective communication between man and computer will coincide with the rise in the intelligence of the computer—so that the human can say more while thinking less. But at this point in history, the only way we can obtain more intelligent machines is to design them—we cannot yet grow them, or breed them, or train them by the blind procedures that work with humans. We are caught at the wrong equilibrium of a bistable system: we could design more intelligent machines if we could communicate to them better; we could communicate to them better if they were more intelligent. Limited both in our capabilities for design and communication, every advance in either separately requires a momentous effort. Each success, however, allows a corresponding effort on the other side to reach a little further. At some point the reaction will "go," and we will find ourselves at the favorable equilibrium point of the system, possessing mechanisms that are both highly intelligent and communicative.

With this view of the task and its setting, we can turn to the substance of the paper: the development of chess programs. We will proceed historically, since this arrangement of the material will show most clearly what progress is being made in obtaining systems of increasing complexity and intelligence.

Shannon's Proposal

The relevant history begins with a paper by Claude Shannon in 1949.[11] He did not present a particular chess program, but discussed many of the basic problems involved. The framework he introduced has guided most of the subsequent analysis of the problem.

As Shannon observed, chess is a finite game. There is only a finite number of positions, each of which admits a finite number of alternative moves. The rules of chess assure that any play will terminate: that eventually a

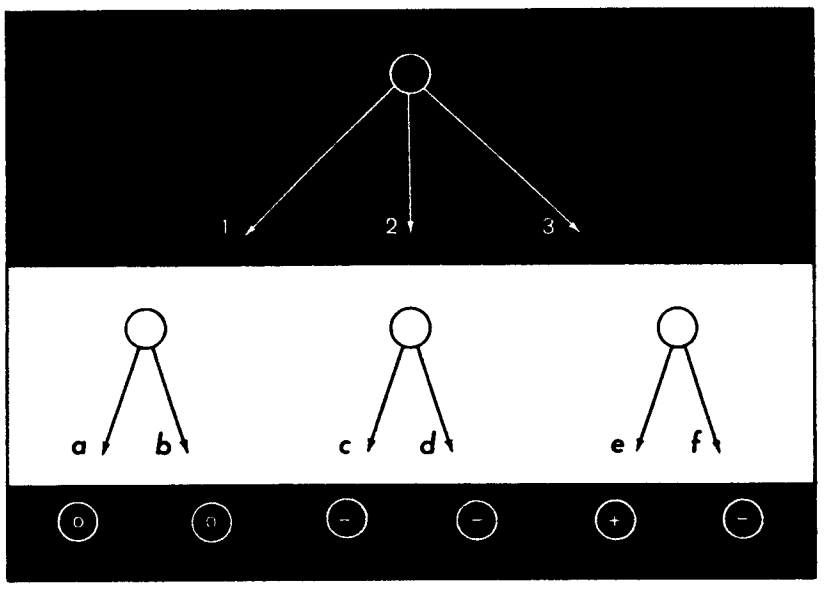


Figure 2 **The game tree and minimaxing.**

position will be reached that is a win, loss, or draw. Thus chess can be completely described as a branching tree (as in Fig. 2), the nodes corresponding to positions and the branches corresponding to the alternative moves from each position. It is intuitively clear, and easily proved, that for a player who can view the entire tree and see all the ultimate consequences of each alternative, chess becomes a simple game. Starting with the terminal positions, which have determinate payoffs, he can work backwards, determining at each node which branch is best for him or his opponent as the case may be, until he arrives at the alternative for his next move.

This inferential procedure—called *minimaxing* in the theory of games—is basic to all the attempts so far to program computers for chess. Let us be sure we understand it. Figure 2 shows a situation where White is to move and has three choices, (1), (2), and (3). White's move will be followed by Black's: (a) or (b) in case move (1) is made; (c) or (d) if move (2) is made; and (e) or (f) if move (3) is made. To keep the example simple, we have assumed that all of Black's moves lead to positions with known payoffs: (+) meaning a win for White, (0) meaning a draw, and (−) meaning a loss for White. How should White decide what to do—what inference procedure allows him to determine which of the three moves is to be preferred? Clearly, no matter what Black does, move (1) leads to a draw. Similarly, no matter what Black does, move (2) leads to a loss for White. White should clearly prefer move (1) to move (2). But what about move (3)? It offers the possibility of a win, but also contains the possibility of a loss; and furthermore, the outcome is in Black's control. If White is willing to impute any analytic ability to his opponent, he must conclude that move (3) will end as a loss for White, and hence that move (1) is the preferred move. The win from move (3) is completely insubstantial, since it can never be realized. Thus White can impute a value to a position—in this case draw—by reasoning backwards from known values.

To repeat: If the entire tree can be scanned, the best move can be determined simply by the minimaxing pro-

cedure. Now minimaxing might have been the "wheel" of chess—with the adventure ended almost before it had started—if the tree were not so large that even current computers can discover only the minutest fraction of it in years of computing. Shannon's estimate, for instance, is that there are something like 10^{120} continuations to be explored, with less than 10^{16} microseconds available in a century to explore them.

Shannon then suggested the following framework. Playing chess consists of considering the alternative moves, obtaining some effective evaluation of them by means of analysis, and choosing the preferred alternative on the basis of the evaluation. The analysis—which is the hard part—could be factored into three parts. First, one would explore the continuations to a certain depth. Second, since it is clear that the explorations cannot be deep enough to reach terminal positions, one would evaluate the positions reached at the end of each exploration in terms of the pattern of men on the chess board. These static evaluations would then be combined by means of the minimaxing procedure to form the effective value of the alternative. One would then choose the move with the highest effective value. The rationale behind this factorization was the reasonableness that, for a given evaluation function, the greater the depth of analysis, the better the chess that would be played. In the limit, of course, such a process would play perfect chess by finding terminal positions for all continuations. Thus a metric was provided that measured all programs along the single dimension of their depth of analysis.

To complete the scheme, a procedure was needed to evaluate positions statically — that is, without making further moves. Shannon proposed a numerical measure formed by summing, with weights, a number of factors or scores that could be computed for any position. These scores would correspond to the various features that chess experts assert are important. This approach gains plausibility from the existence of a few natural quantities in chess, such as the values of pieces, and the mobility of men. It also gains plausibility, of course, from the general use in science and engineering of linearizing assumptions as first approximations.

To summarize: the basic framework introduced by Shannon for thinking about chess programs consists of a series of questions:

1. Alternatives
   Which alternative moves are to be considered?
2. Analysis
   a) Which continuations are to be explored and to what depth?
   b) How are positions to be evaluated statically—in terms of their patterns?
   c) How are the static evaluations to be integrated into a single value for an alternative?
3. Final choice procedure
   What procedure is to be used to select the final preferred move?

We would hazard that Shannon's paper is chiefly remembered for the specific answers he proposed to these questions: consider all alternatives; search all continuations to fixed depth, *n;* evaluate with a numerical sum; minimax to get the effective value for an alternative; and then pick the best one. His article goes beyond these specifics, however, and discusses the possibility of selecting only a small number of alternatives and continuations. It also discusses the possibility of analysis in terms of the functions that chess men perform—blocking, attacking, defending. At this stage, however, it was possible to think of chess programs only in terms of extremely systematic procedures. Shannon's specific proposals have gradually been realized in actual programs, whereas the rest of his discussion has been largely ignored. And when proposals for more complex computations enter the research picture again, it is through a different route.

Turing's Program

Shannon did not present a particular program. His specifications still require large amounts of computing for even such modest depths of analysis as two or three moves. It remained for A. M. Turing[3] to describe a program along these lines that was sufficiently simple to be simulated by hand, without the aid of a digital computer.

In Table 1 we have characterized Turing's program in terms of the framework just defined. There are some additional categories which will become clear as we proceed. The Table also provides similar information for each of the other three programs we will consider.

Turing's program considered all alternatives—that is, all legal moves. In order to limit computation, however, he was very circumspect about the continuations the program considered. Turing introduced the notion of a "dead" position: one that in some sense was stable, hence could be evaluated. For example, there is no sense in counting material on the board in the middle of an exchange of Queens: one should explore the continuations until the exchange has been carried through — to the point where the material is not going to change with the next move. So Turing's program evaluated material at dead positions only. He made the value of material dominant in his static evaluation, so that a decision problem remained only if minimaxing revealed several alternatives that were equal in material. In these cases, he applied a supplementary additive evaluation to the positions reached by making the alternative moves. This evaluation included a large number of factors—mobility, backward pawns, defense of men, and so on—points being assigned for each.

Thus Turing's program is a good instance of a chess-playing system as envisaged by Shannon, although a small-scale one in terms of computational requirements. Only one published game, as far as we know, was played with the program. It proved to be rather weak, for it lost against a weak human player (who did not know the program, by the way), although it was not entirely a pushover. In general its play was rather aimless, and it was capable of gross blunders, one of which cost it the game. As one might have expected, the subtleties of the evaluation function were lost upon it. Most of the numer-

ous factors included in the function rarely had any influence on the move chosen. In summary: Turing's program was not a very good chess player, but it reached the bottom rung of the human ladder.

There is no *a priori* objection to hand simulation of a program, although experience has shown that it is almost always inexact for programs of this complexity. For example, there is an error in Turing's play of his program, because he—the human simulator—was unwilling to consider all the alternatives. He failed to explore the ones he "knew" would be eliminated anyway, and was wrong once. The main objection to hand simulation is the amount of effort required to do it. The computer is really the enabling condition for exploring the behavior of a complex program. One cannot even realize the potentialities of the Shannon scheme without programming it for a computer.

The Los Alamos Program

In 1956 a group at Los Alamos programmed MANIAC I to play chess.[5] The Los Alamos program is an almost perfect example of the type of system specified by Shannon. As shown in the Table, all alternatives were considered; all continuations were explored to a depth of two moves (i.e., two moves for Black and two for White); the static evaluation function consisted of a sum of material and mobility measures; the values were integrated by a minimax procedure,* and the best alternative in terms of the effective value was chosen for the move.

In order to carry out the computation within reasonable time limits, a major concession was required. Instead of the normal chess board of eight squares by eight squares, they used a reduced board, six squares by six squares. They eliminated the Bishops and all special chess moves: castling, two-square Pawn moves in the opening, and *en passant* captures.

The result? Again the program is a weak player, but now one that is capable of beating a weak human player, as the machine demonstrated in one of its three games. It is capable of serious blunders, a common characteristic, also, of weak human play.

Since this is our first example of actual play on a computer, it is worth looking a bit at the programming and machine problems. In a normal 8×8 game of chess there are about 30 legal alternatives at each move, on the average, thus looking two moves ahead brings 30^4 continuations, about 800,000, into consideration. In the reduced 6×6 game, the designers estimate the average number of alternatives at about 20, giving a total of about 160,000 continuations per move. Even with this reduction of five to one, there are still a lot of positions to be looked at. By comparison, the best evidence suggests that a human player considers considerably less than 100 positions in the analysis of a move.[4] The Los Alamos program was able to make a move in about 12 minutes on the average. To do this the code had to be very simple and straightforward. This can be seen by the size of the program—only 600 words. In a sense, the machine barely glanced at each position it evaluated. The two measures in the evaluation function are obtained directly from the process of looking at continuations: changes in material are noticed if the moves are captures, and the mobility score for a position is equal to the number of new positions to which it leads—hence is computed almost without effort when exploring all continuations.

The Los Alamos program tests the limits of simplification in the direction of minimizing the amount of information required for each position evaluated, just as Turing's program tests the limits in the direction of minimizing the amount of exploration of continuations. These programs, especially the Los Alamos one, provide real anchor points. They show that, with very little in the way of complexity, we have at least entered the arena of human play—we can beat a beginner.

Bernstein's Program

Over the last two years Alex Bernstein, a chess player and programmer at IBM, has constructed a chess-playing program for the IBM 704 (for the full 8×8 board).[1,2] This program has been in partial operation for the last six months, and has now played one full game plus a number of shorter sequences. It, too, is in the Shannon tradition, but it takes an extremely important step in the direction of greater sophistication: only a fraction of the legal alternatives and continuations are considered. There is a series of subroutines, which we can call plausible move generators, that propose the moves to be considered. Each of these generators is related to some feature of the game: King safety, development, defending own men, attacking opponent's men, and so on. The program considers at most seven alternatives, which are obtained by operating the generators in priority order, the most important first, until the seven are accumulated.

The program explores continuations two moves ahead, just as the Los Alamos program did. However, it uses the plausible move generators at each stage, so that, at most, 7 direct continuations are considered from any given position. For its evaluation function it uses the ratio of two sums, one for White and one for Black. Each sum consists of four weighted factors: material, King defense, area control, and mobility. The program minimaxes and chooses the alternative with the greatest effective value.

The program's play is uneven. Blind spots occur that are very striking; on the other hand it sometimes plays very well for a series of moves. It has never beaten anyone, as far as we know; in the one full game it played it was beaten by a good player,[1] and it has never been pitted against weak players to establish how good it is.

Bernstein's program gives us our first information about radical selectivity, in move generation and analysis. At 7 moves per position, it examines only 2,500 final positions two moves deep, out of about 800,000 legal continuations. That it still plays at all tolerably with a reduction in search by a factor of 300 implies that the selection mechanism is fairly effective. Of course, the

*The minimax procedure was a slight modification of the one described earlier, in that the mobility score for each of the intermediate positions was added in.