The Game of Chess

Herbert A. Simon
Carnegie-Mellon University, and
Jonathan Schaeffer
University of Alberta

CHESS.MSS

The Game of Chess

Herbert A. Simon
Carnegie-Mellon University and
Jonathan Schaeffer
University of Alberta

The game of chess has sometimes been referred to as the Drosophila of artificial intelligence and cognitive science research -- a standard task that serves as a test bed for ideas about the nature of intelligence and computational schemes for intelligent systems. Both machine intelligence -- how to program a computer to play good chess (artificial intelligence) -- and human intelligence -- how to understand the processes that human masters use to play good chess (cognitive science) -- are encompassed in the research, and we will comment on both in this chapter, but with emphasis on computers.

From the standpoint of von Neumann-Morgenstern game theory [14], chess may be described as a trivial game. It is a two-person, zero-sum game of perfect information. Therefore the rational strategy for play is obvious: follow every branch in the game tree to a win, loss, or draw -- the rules of the game guarantee that only a finite number of moves is required. Assign 1 to a win, 0 to a draw, and -1 to a loss, and minimax backwards to the present position.

For simple games, such as tic-tac-toe and cubic, the search space is small enough to be easily exhausted, and the games are readily solved by computer. Recently, the game of connect-four, with a search space of 10^{13} positions, was solved; with perfect play, the player moving first (white) will always win [13]. This result was not obtained by a full search of the space, but by discovering properties of positions that, whenever present in a position, guarantee a win. In this way, large sub-trees of the game tree could be evaluated without search.

The only defect in trying to apply this optimal strategy to chess is that neither human beings nor the largest and fastest computers that exist (or that are in prospect) are capable of executing it. The number of legally possible games of chess is estimated roughly at 10^{44}, a number that probably exceeds the number of molecules in the universe. An exploration of this magnitude is not remotely within reach of achievable computing devices, human or machine, now or in the future.

In the literature of economics, the distinction has sometimes been made between *substantive rationality* and *procedural (a.k.a. computational) rationality*. Substantive rationality is concerned with the objectively correct or best action, given the goal, in the specified situation.

Classical game theory has been preoccupied almost exclusively with substantive rationality.

Procedural rationality is concerned with procedures for finding good actions, taking into account not only the goal and objective situation, but also the knowledge and the computational capabilities and limits of the decision maker. The only non-trivial theory of chess is a theory of procedural rationality in choosing moves. The study of procedural or computational rationality is relatively new, having been cultivated extensively only since the advent of the computer (but with precedents, e.g., numerical analysis). It is central to such disciplines as artificial intelligence and operations research.

Difficulty in chess, then, is computational difficulty. Playing a good game of chess consists in using the limited computational power (human or machine) that is available to do as well as possible. This might mean investing a great deal of computation in examining a few variations, or investing a little computation in each of a large number of variations. Neither strategy can come close to exhausting the whole game tree -- to achieving substantive rationality.

Human Chess Play

To get an initial idea of how the task might be approached, we can look at what has been learned over the past half century about human chess play, which has been investigated in some depth by a number of psychologists. There is a considerable understanding today about how a grandmaster wins games -- not enough understanding, alas, to make it easy to become a grandmaster.

First, since the pioneering studies of the Dutch psychologist, Adriaan de Groot, we have known that a grandmaster, even in a difficult position, carries out a very modest amount of search of the game tree -- probably seldom more than 100 branches [6]. Even if this is an underestimate by an order of magnitude (it probably is not), 10^3 is a miniscule number compared with 10^{44}. In fact, de Groot found it very difficult to discriminate, from the statistics of search, between grandmasters and ordinary club players. The only reliable difference was that the grandmasters consistently searched more relevant variations and found better moves than the others -- not a very informative result.

It should not surprise us that the skilled chess player makes such a limited search of the possibilities. The human brain is a very slow device (by modern electronic standards). It takes about a millisecond for a signal to cross a single synapse in the brain, hence ten to a hundred milliseconds for anything "interesting" to be accomplished. In a serious chess game, a player must make moves at the rate of twenty or thirty per hour, and only ten minutes or a quarter hour can be allotted to even a difficult move. If 100 branches in the game tree were examined in fifteen minutes, that would allow only nine seconds per branch -- not a large amount of time for a system that operates at human speeds.

A second thing we know is that when grandmasters look at a chessboard in the course of a game, they see many familiar patterns or "chunks" -- patterns they have often seen before in the course of play. Moreover, they not only immediately notice and recognize these chunks whenever they encounter them, but they have available in memory information that gives the chunks significance -- tells what opportunities and dangers they signal, what moves they suggest. The capability for recognizing chunks is like a very powerful index to the grandmaster's encyclopedia of chess. When the chunk is recognized, the relevant information stored with it is evoked and recalled from memory.

The grandmaster's vast store of chunks seems to provide the main explanation for the ability to play many simultaneous games rapidly. Instead of searching the game tree, the grandmaster simply makes "positionally sound" moves until the opponent makes an inferior move, creating a (usually slight) weakness. The grandmaster at once notices this clue and draws on the associated memory to exploit the opponent's mistake.

Rough estimates place the grandmaster's store of chunks of chess knowledge at a minimum of 50,000 -- perhaps even twice that or more. This number is comparable in magnitude to the typical native language vocabularies of college-educated persons.

There is good evidence that it takes a minimum of ten years of intense application to reach a strong grandmaster level in chess. Even prodigies like Bobby Fischer have required that. (The same period of preparation is required for world class performance in all of the dozen other fields that have been examined.) Presumably, a large part of this decade of training is required to accumulate and index the 50,000 chunks.

It has sometimes been supposed that the grandmaster not only has special knowledge, but also special cognitive capabilities, especially the ability to grasp the patterns on chess boards in mental images. An interesting and simple experiment, which has been carried out by several investigators and is easily replicated, makes very clear the nature of chess perception. (The analogous experiment has been performed with other games, like bridge, with the same result.)

Allow a subject in the laboratory to view a chess position from a well-played game (with perhaps 25 pieces on the board) for five to ten seconds. Then remove the pieces and ask the subject to reconstruct the position. If the subject is a master or grandmaster, 90% or more of the pieces (23 out of 25, say) will be replaced correctly. If the subject is an ordinary player, only about six will be replaced correctly. This is an enormous and startling difference between excellent and indifferent chess players. Something about their eyes?

Next repeat the experiment, but place the same pieces on the board *at random*. The ordinary player will again replace about six on the right squares. Now the master or grandmaster will also only replace about six correctly. Clearly the expert’s pre-eminence in the first version of the experiment has nothing to do with visual memory -- it has to do with familiarity. A chessboard from a well-played game contains 25 nearly unrelated pieces of information for the ordinary player, but only a half dozen familiar patterns for the master. The pieces group themselves for the expert into a small number of interrelated chunks.

Few familiar chunks will appear on the randomly arranged board, hence, in remembering that position, the expert will face the same 25 unrelated pieces of knowledge as the novice. Evidence from other domains shows that normal human adults can hold about six or seven unrelated items in short-term memory at the same time (equivalent to one unfamiliar telephone number). There is nothing special about the chess master’s eyes, but a great deal that is special about his or her knowledge -- the indexed encyclopedia stored in memory.

In summary, psychological research on chess thinking shows that it involves a modest amount of search in the game tree (a maximum of 100 branches, say) combined with a great deal of pattern recognition, drawing upon patterns stored in memory as a result of previous chess training and experience. The stored knowledge is used to guide search in the chess tree along the most profitable lines, and to evaluate the terminal positions that are reached in the search. These estimated values at the terminals of the miniature game trees are what are minimaxed (if anything is) in order to select the best move. For the grandmaster, vast amounts of chess knowledge compensate for the very limited ability of the slow human neurological "hardware" to conduct extensive searches in the time available for a move.

Computer Chess: Origins

The idea of programming computers to play chess appeared almost simultaneously with the birth of the modern computer (see [9] for a detailed history to 1970 and [15] for a more modern perspective). Claude Shannon, the creator of modern information theory, published a proposal for a computer chess program in 1950, and the score of a game played by a hand-simulated program devised by A. M. Turing was published in 1953. A substantial number of other designs were described and actually programmed and run in the succeeding decade.

There was a close family resemblance among most of the early programs. The task was viewed in a game-theoretic framework. Alternative moves were to be examined by search through the tree of legal moves, values were to be assigned to terminal nodes on the tree, and the values were to be minimaxed backwards through the tree to determine values for the initial moves. The move with the largest value was then selected and played. The same search and evaluation program represented both the player and the opponent.

Of course the programs represented only the crudest approximations to the optimal strategy of von Neumann-Morgenstern game theory. Because of the severe computational limits (and the early programs could examine at most a few thousand branches of the tree), a function for evaluating the (artificial) terminals had to be devised that could approximate the true game values of the positions at these terminals.

This evaluation function was, and remains, the most vulnerable Achilles heel in computer chess. We shall see that no one, up to the present time, has devised an evaluation function that does not make serious mistakes from time to time in choosing among alternative positions, and minor mistakes quite frequently. Although the best current chess programs seldom blunder outright, they often make moves that masters and grandmasters rightly regard as distinctly inferior to the best moves.

In addition to brute force search, two other ideas soon made their appearance in computer chess programs. The first (already proposed by Shannon in 1950) was to search selectively, rather than exhaustively, sacrificing completeness in the examination of alternatives in order to attain greater depth along the lines regarded as most important. Of course it could not be known with certainty which lines these were; the rules of selection were heuristic, rules of thumb that had no built-in guarantees of correctness. These rules could be expressed in terms of position evaluation functions, not necessarily identical with the function for evaluating terminal nodes.

Turing made the important distinction between "dead" positions, which could reasonably be evaluated in terms of their static features, and "live" positions having unresolved dynamic features (pieces *en prise*, and the like) that required further search before they could be evaluated. This distinction was incorporated in most subsequent chess programs.

The second departure from the basic game-theoretic analogue was to seek ways of reducing the magnitude of the computation by examining branches in the best order. Here there arose the idea of alpha-beta search, which dominated computational algorithms for chess during the next three decades.

The idea underlying alpha-beta search is roughly this. Suppose that, after a partial search of the tree from node A, a move, M1, has already been found that guarantees the player at A the value V. Suppose that after a partial search of one of the other moves at A, M2, a reply has been found for the opponent that guarantees him or her a value (for the player at A) of less than V. Then there is no need to carry on further search at the subnode, as M2 cannot be better than M1. Roughly speaking, use of this heuristic reduces the amount of computation required to the square root of the amount without the heuristic, a substantial reduction. The alpha-beta heuristic, which has many variant forms, is a form of the branch-and-bound heuristic widely used in solving various combinatorial problems of management science.

While the alpha-beta heuristic provides a quite powerful tree-pruning principle, when it is combined (as it must be in chess) with an evaluation function that is only approximate, it is not without its subtleties and pitfalls. Nau first demonstrated that minimax trees with approximate evaluation may be *pathological*, in that deeper search may lead to poorer decisions [8]. This apparent paradox is produced by the accumulation of errors in evaluation. Although this important result has been ignored in the design of chess programs, pathology probably does occur in tree search, but not to the extent of degrading performance seriously.

The dead position heuristic, alpha-beta search, and the other selective heuristics that served to modify brute-force game-theoretic minimaxing were all responses to the devastating computational load that limited depth of search, and the inability to devise, for these shallow searches, an evaluation function that yielded a sufficiently close approximation to the true value of the terminal positions. We must remember that during the first decade or two of this research, computers were so small and slow that full search could not be carried to depths of more than about two moves (four or five ply, or an average of about 1,000 to 30,000 branches).

We might mention two programs, one of 1958, the other of 1966, that departed rather widely from these general norms of design. Both were planned with the explicit view of approximating, more closely than did the game-theory approximation, the processes that human players used to choose moves.

The first of these programs (Newell, Shaw and Simon, 1958) had separate base-move generators and analysis-move generators for each of a series of goals: material balance, center control, and so on [9]. The moves proposed by these generators were then evaluated by separate analysis procedures that determined their acceptability. The program, which was never developed beyond three goals for the opening, played a weak game of chess, but demonstrated that, in positions within its limited scope of knowledge, it could find reasonable moves with a very small amount of search (much less than 100 branches.)

Another important feature of the NSS program was the idea of *satisficing*, that is, choosing the first move that was found to reach a specified value. In psychological terms, this value could be viewed as a level of aspiration, to be assigned on the basis of a preliminary evaluation of the current position, with perhaps a small optimistic upward bias to prevent too early termination of search. Satisficing is a powerful heuristic for reducing the amount of computation required, with a possible sacrifice of the quality of the moves chosen -- but not necessarily a sacrifice when total time constraints are taken into account.

Another somewhat untypical program, MATER, was specialized to positions where the player has a tactical advantage that makes it profitable to look for a checkmating combination [9]. By giving priority to forceful moves, and looking first at variations that maximally restricted the number of legal replies open to the opponent, the program was able to find very deep mating combinations (up to eight moves, 15 ply, deep) with search trees usually well under 100 branches.

An interesting successor to this program, which combines search and knowledge, with emphasis on the latter, is Wilkins' program PARADISE [16]. PARADISE built small "humanoid" trees, relying on extensive chess knowledge to guide its search. Within its domain it was quite powerful. However, like MATER, its capabilities were limited to tactical chess problems (checkmates and wins of material), and it could not perform in real time.

During this same period, stronger programs that adhered more closely to the "standard" design described earlier were designed by Kotok, Greenblatt, and others [9]. Gradual improvements in strength were being obtained, but this was due at least as much to advances in the sizes and speeds of computer hardware as to improvements in the chess knowledge provided to the programs or more efficient search algorithms.

Search versus Knowledge

Perhaps the most fundamental and interesting issue in the design of chess programs is the trade-off between search and knowledge. As we have seen, human players stand at one extreme, using extensive knowledge to guide a search of perhaps a few hundred positions and then to evaluate the terminal branches of the search tree. Brute-force chess programs lie at the opposite extreme, using much less knowledge but considering millions of positions in their search. Why is there this enormous difference?

We can view the human brain as a machine with remarkable capabilities, and the computer, as usually programmed, as a machine with different, but also remarkable capabilities. In the current state of computer hardware and software technology, computers and people have quite different strengths and weaknesses. Most computer programs lack any capacity for learning, while human brains are unable to sum a million numbers in a second. When we are solving a problem like chess on a computer, we cater to the strengths rather than the weaknesses of the machine. Consequently, chess programs have evolved in a vastly different direction from their human counterparts. They are not necessarily better or worse; just different.

Search

Progress in computer chess can be described subjectively as falling into three epochs, distinguished by the search methods the programs used. The pre-1975 period, described above, could be called the *pioneering era*. Chess programs were still a novelty and the people working on them were struggling to find a framework within which progress could be made (much like Go programmers today). By today's standards, many of the search and knowledge techniques used in the early programs were *ad hoc*, with some emphasis on selective, knowledge-based search. Some (not all) programs sought to emulate the human approach to chess, but this strategy achieved only limited success.

The 1975-1985 period could be called the *technology era*. Independently, several researchers discovered the benefits of depth-first, alpha-beta search with iterative deepening. This was a reliable, predictable and easily implemented procedure. At the same time, a strong correlation was observed between program speed (as measured by positions considered per second) and program performance. Initially, it was estimated that an additional ply of search (increasing the depth of search by a move by White or Black) could improve performance by as much as 250 rating points.[1] At this rate, overcoming the gap between the best programs (1800) of the middle 1970s and the best human players (2700) required only 4 ply of additional search. Since an extra ply costs roughly a factor of 4-8 in computing power, all one had to do was wait for technology to solve the problem by producing computers 4,000 times faster than those then available -- something that would surely be achieved in a few years.

Unfortunately, the rating scale is not linear in amount of search, and later experience indicated that beyond the master level (2200 points), each ply was worth only an additional 100 points. And, of course, it is widely believed that the rate of gain will decrease even further as chess programs reach beyond the base of the grandmaster (2500) level.

All the top programs today reflect this fascination with brute-force alpha-beta, having an insatiable appetite for speed. The major competitive chess programs run on super-computers (*Cray Blitz*), special purpose hardware (*Belle*, *Hitech*, *Deep Thought*) and multi-processors (*Phoenix*). The best chess programs reached the level of strong masters, largely on the coat tails of technology rather than by means of major innovations in software or knowledge engineering.

Since 1985, computer chess has hit upon a whole host of new ideas that are producing an emerging *algorithm era*. The limits of efficiency in alpha-beta search had been reached; new approaches were called for [12]. In quick succession, a number of innovative approaches to search appeared. Whereas traditional alpha-beta search methods examine the tree to a fixed depth (with some possible extensions), the new approaches attempt to expand the tree in places where additional effort is likely to produce a more accurate value at the root. Alpha-beta relies on depth as the stopping criterion; removal of this restriction is the most significant aspect of the new ideas in search.

Alpha-beta returns the maximum score achievable (relative to the fallible evaluation function) and a move that achieves this score. Little information is provided on the quality of other sibling moves. *Singular extensions* is an enhancement to alpha-beta to perform additional searches to determine when the best move in a position is *significantly* better than all the alternatives [1]. These *singular* or *forced* moves are then re-searched deeper then they normally would be. Consequently, a forcing sequence of moves will be searched to a greater depth than

[1]Chess performance is usually measured on the so-called Elo scale, where a score of 2,000 represents Expert performance, 2,200 Master level performance, and 2,500 Grandmaster performance. There are both American and International chess ratings, which differ by perhaps 100 points, but the above approximation will be sufficient for our purposes.

with conventional alpha-beta.

The *conspiracy numbers* algorithm maintains a count of the number of terminal nodes in the tree that must change value (or *conspire*) to cause a change in the root value [7]. Once the number of conspirators required to cause a certain change in the root exceeds a prescribed *threshold*, that value is considered unlikely to occur and the corresponding move is removed from consideration. The search stops when one score for the root is *threshold* conspirators better than all other possible scores.

Min/max approximation uses mean value computations to replace the standard minimum and maximum operations of alpha-beta [10]. The advantage of using mean values is that they have continuous derivatives. For each terminal node, a derivative is computed that measures the sensitivity of the root value to a change in value of that node. The terminal node that has the most influence on the root is selected for deeper searching. The search terminates when the influence on the root of all terminal nodes falls below a set minimum. Note that when min/max approximation and conspiracy numbers are used, alpha-beta cut-offs are not possible.

Equi-potential search expands all terminal nodes with a marginal utility greater than a prescribed threshold [2]. The utility of a node is a function of the search results known at all interior nodes along the path from the tree root to that node, together with any useful heuristic information provided by the knowledge of the program. The search terminates when all terminal nodes have a marginal utility below the prescribed threshold.

In addition to these procedures, a combination of selective search with brute-force search has re-emerged as a viable strategy.

Although many of these ideas have yet to make the transition from theory to practice, they are already strongly influencing the directions today's programs are taking. In some sense, it is unfortunate that computer chess was given the gift of alpha-beta search so early in its infancy. Although it was a valuable and powerful tool, its very power may have inadvertently slowed progress in chess programs for a decade.

Knowledge

In view of the obvious importance of evaluation functions, it may seem surprising that more effort has not been devoted to integrating knowledge in a reliable manner into chess programs. The reason is simple: program performance is more easily enhanced through increases in speed (whether hardware or software) than through knowledge acquisition. With present techniques, capturing, encoding and tuning chess knowledge is a difficult, time consuming and *ad hoc* process. Because performance has been the sole metric by which chess programs were judged, there has been little incentive for solving the knowledge problem.

Acquiring knowledge by having human grandmasters advise chess programmers on the weaknesses of their programs has not proved effective: the two sides talk different languages. Consequently, few chess programs have been developed in consultation with a strong human player.

Deeper search has had the unexpected effect of making chess programs appear more knowledgeable than they really are. As a trivial example, consider a chess program that has no knowledge of the concept of *fork*, a situation in which a single piece threatens two of the opponent's pieces simultaneously. Searching a move deeper reveals the captures without the need for representing the fork in the evaluation of the base position. In this manner deep searches can compensate for the absence of some (important) kinds of knowledge, by detecting and investigating the effects of the unnoticed features.

Arthur Samuel constructed, in the 1950s, a powerful program for playing checkers [11]. The program's knowledge was embodied in a complex evaluation function (which was capable of self-improvement through learning processes), and the program's prowess rested squarely on the accuracy of its evaluations. The published descriptions of the program provide a complete description of the checkers knowledge incorporated in it. (Comparable detailed information is not available for most chess programs.) However, Samuel's program is now three decades old, and technology has improved machine speed by several orders of magnitude. Checkers programs today use less than half of the knowledge of Samuel, the programs depending on search to reveal the rest. The same holds true for chess programs.

That skilled human players process chess positions in terms of whole groups of pieces, or chunks, is recognized as important by chess programmers; but, with few exceptions, no one has shown how to acquire or use chunks in programs, particularly under the real-time constraints of tournament chess games. One of the exceptions is Campbell's CHUNKER program, which is able to group the pieces in king and pawn endgames and reason about the chunks in a meaningful way [5]. Although the search trees built by CHUNKER are larger than would be built by a skilled human player, CHUNKER, in the limited class of positions it could handle, achieved a level of performance comparable to that of a grandmaster.

Only recently has significant effort in the design of chess programs been re-directed towards the acquisition and use of chess knowledge. Knowledge is useful not only for position evaluation, but also to guide the direction of search effort. The *Hitech* chess program, one of the two or three strongest programs in existence today, employs high-speed, special-purpose parallel hardware, but also incorporates more complete and sophisticated chess knowledge than any other program built to date [4]. Over a three year period, and without any change in hardware to increase its speed, the program improved in strength from an expert to a strong master solely on the basis of software improvements, principally to, and improvements of, its chess knowledge.

A Tale of Two Programs

It is interesting to compare the current two best chess programs. Both originate at Carnegie-Mellon University and both use special-purpose VLSI processors, but that is where the similarities end. *Deep Thought* concentrates on speed; a specially designed, single chip chess machine that can analyze 500,000 positions per second. Further, since the basic design is easily reproducible, one can have multiple copies running in parallel, achieving even better performance (the system currently uses 4). As a consequence, under tournament conditions the program can look ahead at least 2 moves (or ply) deeper than any other program. However, considerable work remains to be done with its software to allow the program to overcome the many glaring gaps in its chess knowledge.

Hitech also uses special-purpose chips; 64 of them, one for each square on the board. However, *Hitech*'s speed is less than that of a single *Deep Thought* processor. *Hitech*'s strength lies its knowledge base. The hardware has been enhanced to include pattern recognizers that allow the program to represent and manipulate knowledge at a level not possible in *Deep Thought*. Consequently, *Hitech* plays a a more "human-like" game of chess, without many of the "machine" tendencies that usually characterize computer play. However, *Deep Thought*'s tremendous speed allows it to search the game tree to unprecedented depths, often finding unexpected hidden resources in positions that take both other computer programs and human players by surprise.

A game between *Hitech* and *Deep Thought* is analogous to a boxing batch between fighters with quite different styles. *Hitech* has the greater finesse, and will win most rounds; whereas *Deep Thought* has the knock-out punch. Unfortunately for finesse, no one remembers that you out-boxed your opponent for ten rounds. All they remember is that you were knocked out in the eleventh.

Computer Chess Play

Unfortunately, the current search-intensive approaches, using minimal chess knowledge, have not been able to eliminate glaring weaknesses from the machines' play. Now that machines can wage a strong battle against the best human players, their games are being studied and their weaknesses uncovered. Chess masters who are given the opportunity to examine the games of future opponents are very good at identifying and exploiting weaknesses. Since existing computer programs do not improve their play through learning[2], they are inflexible, unable to understand or prevent the repeated exploitation of a weakness perceived by an opponent.

Learning, in the current practice of computer chess, consists of the programmer observing the kinds of trouble the program gets into, identifying the causes, and then modifying the program to remove the problem. The programmer, not the program, does the learning!

The biggest shortcomings of current chess programs undoubtably lie in their knowledge bases. Programs are too quick to strive for short-term gains visible in their relatively shallow search trees, without taking into account the long-term considerations that lie beyond the horizons of the search. The absence of fundamental knowledge-intensive aspects of human chess play, such as strategic planning, constitute a major deficiency in chess programs. In addition, they are unable to learn from mistakes and to reason from analogies, so that they cannot improve dynamically from game to game.

On the other hand, chess programs are not susceptible to important human weaknesses. The ability to calculate deeply and without computational (as opposed to evaluational) errors are enormous advantages. They are also free from psychological problems that humans have to overcome. Humans, tuned to playing other humans, are frequently flustered by computer moves. Machines do not subscribe to the same preconceptions and aesthetics that humans do, and often discover moves that catch their opponents off guard. Many games have been lost by human opponents because of over-confidence induced by the incorrect perception that the machine's moves were weak.

Further, a computer has no concept of fear and is quite content to walk along a precipice without worry about the consequences of error. A human, always mindful of the final outcome of

[2] Other than to remember previous games played, so that they can avoid repeating the same mistake in the same opening position (but not in similar positions).

the game, can be frightened away from his best move because of fear of the unknown and the risk of losing.

Only recently have people questioned whether the human model of how to play chess well is the only good model or, indeed, even the best model. The knowledgeable chess player is indoctrinated with over 100 years of chess theory and practice, leaving little opportunity for breaking away from conventional thinking on how to play chess well. On the other hand, a computer program starts with no pre-conceptions. Increasingly, the evidence suggests that computer evaluation functions, combined with deep searches, may yield a different model of play that is *better* than the one humans currently use!

In fact, it is quite likely that computer performance is inhibited by the biases in the human knowledge provided to programs by the programmer. Since a machine selects moves according to different criterion than those used by people (these criterion may have little relation to anything a chess player "knows" about chess), it is not surprising that the machines' styles of play are "different" and that humans have trouble both in playing against them and in improving their programs.

The Future

In 1988, *Deep Thought* became the first chess program to defeat a grandmaster in a serious tournament game. Since then, it has been shown that this wasn't a fluke, for two more grandmasters have fallen. *Deep Thought* is acknowledged to be playing at the International Master level (2400+ rating) and *Hitech* is not far behind.

However, in 1989, World Champion Garry Kasparov convincingly demonstrated that computers are not yet a threat for the very best players. He studied all of *Deep Thought*'s published games and gained insight into the strengths and weaknesses of the program's play. In a two-game exhibition match, Kasparov decisively beat *Deep Thought* in both games.

It is difficult to predict when computers will defeat the best human player. This important event, so long sought since the initial optimism of Newell and Simon (1957) was disappointed, will be a landmark in the history of artificial intelligence. With the constantly improving technology, and the potential for massively parallel systems, it is not a question "if" this event will occur but rather "when". As the brute-force programs search ever more deeply, the inadequacy of their knowledge is overcome by the discoveries made during search. It can only be a matter of a few years before technological advances end the human supremacy at chess.

Currently, computer chess research appears to be moving in two divergent directions. The first continues the brute-force approach, building faster and faster alpha-beta searchers. *Deep Thought*, with its amazing ability to consider 2,000,000 chess positions per second, best epitomizes this approach.

The second direction is a more knowledge-intensive approach. *Hitech* has made advances in this direction, combining extensive chess knowledge with fast, brute-force search. Commercial manufacturers of chess computers, such as *Mephisto*, realized long ago that they could never have the hardware to compete with the powerful research machines. To compensate for their lack of search depth (roughly 10,000 positions per second), they have devoted considerable attention to applying knowledge productively. The results have been impressive.

Which approach will win out? At the 1989 North American Computer Chess Championship, *Mephisto*, running on a commercially available micro-processor, defeated *Deep Thought*. The verdict is not in.

Conclusion

Many of the lessons learned from writing computer chess programs have been substantiated by experience in constructing programs for other games: bridge bidding and play, Othello, backgammon, checkers, and Go, to mention a few. In both backgammon and Othello, the best programs seem to match or exceed the top human performances. Backgammon is especially interesting since the interpolation of moves determined by the throw of dice makes tree search almost futile. As the basis for evaluating and choosing moves, programs rely on pattern recognition, as do their human counter-parts, and exact probability calculations, for which the human usually approximates or uses intuition. Berliner's construction of a world-class backgammon program using pattern recognition [3] paved the way for building a rich body of chess knowledge into *Hitech*.

We have seen that the theory of games that emerges from this research is quite remote in both its concerns and its findings from von Neumann Morgenstern theory. To arrive at actual strategies for the play of games as complex as chess, the game must be considered in extensive form, and its characteristic function is of no interest. The task is not to characterize optimality or substantive rationality, but to define strategies for finding good moves -- procedural rationality.

Two major directions have been explored. On the one hand, one can replace the actual game by a simplified approximation, and seek the game-theoretical optimum for the approximation -- which may or may not bear any close resemblance to the optimum for the real game. In a game like chess, usually it does not. On the other hand, one can depart more widely from exhaustive minimax search in the approximation and use a variety of pattern-recognition and selective search strategies to seek satisfactory moves.

Both of these directions produce at best satisfactory, not optimal, strategies for the actual game. There is no a priori basis for predicting which will perform better in a domain where exact optimization is computationally beyond reach. The experience thus far with chess suggests that a combination of the two may be best -- with computers relying more (but not wholly) on speed of computation, humans relying much more on knowledge and selectivity.

What is emerging, therefore, from research on games like chess, is a computational theory of games: a theory of what it is reasonable to do when it is impossible to determine what is best -- a theory of bounded rationality. The lessons taught by this research may be of considerable value for understanding and dealing with situations in real life that are even more complex than the situations we encounter in chess -- in dealing, say, with large organizations, with the economy, or with relations among nations.

References

[1] T. Anantharaman, M.S. Campbell and F.H. Hsu.
Singular Extensions: Adding Selectivity to Brute-Force Searching.
Artificial Intelligence , 1990.
To appear. Also published in the Journal of the International Computer Chess Association, vol. 4 (1988), pp. 135-143, and in the AAAI Spring Symposium on Game Playing (1987), pp. 8-13.

[2] T. Anantharaman.
A Statistical Study of Selective Min-Max Search.
PhD thesis, Carnegie-Mellon University, 1990.
To appear.

[3] H.J. Berliner.
Computer Backgammon.
Scientific American , June, 1980.

[4] H.J. Berliner and C. Ebeling.
Pattern Knowledge and Search: The SUPREM Architecture.
Artificial Intelligence 38(2):161-198, 1989.

[5] M.S. Campbell and H.J. Berliner.
Using Chunking to Play Chess Pawn Endgames.
Artificial Intelligence 23(2):97-120, 1984.

[6] A.D. de Groot.
Thought and Choice in Chess.
Mouton, The Hague, 1965.

[7] D.A. McAllester.
Conspiracy Numbers for Min-Max Search.
Artificial Intelligence 35(3):287-310, 1988.

[8] D.S. Nau.
Pathology in Game Trees Revisited and an Alternative to Minimaxing.
Artificial Intelligence 21(1-2):221-244, 1983.

[9] A. Newell, H.A. Simon and J. Shaw.
Human Problem Solving.
Prentice-Hall, Princeton, 1972.

[10] R.L. Rivest.
Game Tree Searching by Min/Max Approximation.
Artificial Intelligence 34(1):77-96, 1988.

[11] A.L. Samuel.
Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress.
IBM Journal of Research and Development 11(6):601-617, 1967.

[12] J. Schaeffer.
The History Heuristic and Alpha-Beta Search Enhancements in Practice.
IEEE Transactions on Pattern Analysis and Machine Intelligence 11(11):1203-1212, 1989.

[13] J.W.J.M. Uiterwijk, H.J. van den Herik and L.V. Allis.
A Knowledge-Based Approach to Connect-Four. The Game is Over: White to Move Wins!
Heuristic Programming in Artificial Intelligence.
Ellis Horwood Ltd. (UK), John Wiley (US), 1989.

[14] J. von Neumann and O. Morgenstern.
Theory of Games and Economic Behavior.
Princeton University Press, Princeton, 1944.

[15] D. Welsh and B. Baczynskyj.
Computer Chess II.
W.C. Brown Co., 1985.

[16] D.E. Wilkins.
Using Knowledge to Control Tree Searching.
Artificial Intelligence 18(1):1-51, 1982.