SOME NECESSARY CONDITIONS FOR A MASTER CHESS PROGRAM *

Hans J. Berliner
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania

Abstract

Since 1967 there has again been great interest in chess programming. This paper demonstrates that the structure of today's most successful programs cannot be extended to play Master level chess. Certain basic requirements of a Master player's performance are shown to be outside the performance limits to which a program of this type could be extended. The paper also examines a basic weakness in the tree-searching model approach when applied to situations that cannot be searched to completion. This is the Horizon Effect, which causes unpredictable evaluation errors due to an interaction between the static evaluation function and the rules for search termination. The outline of a model of chess playing that avoids the Horizon Effect and appears extendable to play Master level chess is presented, together with some results already achieved.

KEYWORDS: Game playing, chess programming, heuristic programming, Horizon Effect.

Introduction

Since 1967 chess programming has again been an interesting subject for Artificial Intelligence researchers. At that time Greenblatt [5] developed a program which soon proved to play nearly as well as the mid-range of registered human players in America.

The basic structure of Greenblatt's program can be historically traced through what Shannon [10] in 1950 called a Type B strategy, an initial effort by Bernstein et. al. [3] in 1958, another more successful program documented in Kotok [6] 1962, and finally Greenblatt. Since then there have been several other programs of this type developed. Of these only the Northwestern University program [1] has achieved a human Class C rating, which places it in a unique position along with the Greenblatt program.

Briefly, the type of program being discussed searches to a depth of five ply under tournament conditions (in endgames with very few legal moves they may search from one to three ply deeper). They do not investigate every legal move, but restrict themselves to a sub-set at each node. Moves are selected for further search on the basis of a scoring function which attempts to rank order the legal moves according to their goodness. The number of moves selected for further examination is usually a function of the depth at which the node occurs. In special situations there are mechanisms which allow the search to be expanded beyond this pruned set, in order to attempt to meet unanticipated problems.

* This work was supported by the Advanced Research Projects Agency, Department of Defence, under contract number F44620-70-C-0107.

Programs of this type typically evaluate between 5,000 and 50,000 bottom nodes in the trees they generate. They do this by applying a static evaluation function, which, since it must be invoked so many times in the course of the three minutes allowed for a tournament move, must of necessity be restricted to a few milliseconds of computation.

Because programs exist which can compete in human tournaments, and because of the annual computer competition et the ACM meeting, there now exists considerable interest all over the world in the future of computer chess. In fact, there is a detectable expectancy in the AI community that a Master level chess program will exist before the end of the decade.

It is the purpose of this paper to show that a program with the structure of today's most successful programs cannot be extended to play Master chess. When one considers that much effort by several groups has only succeeded in raising the 1968 standard of performance of the Greenblatt program by a hardly measurable amount, there is some reason to believe that the present design is already near the assymptote of its potential. We will show that the domain in which a Master level program has to operate could never be subsumed in any domain to which the above programs could be extended. We will also show that certain features in the evaluation procedure used by current programs lead to basic errors that can not be tolerated in Master play. Finally, the outline of a model of chess that could perform as required Is presented.

We will draw examples from both human and machine play. In order to minimize the chess knowledge required of the reader, the examples have been chosen to be as obvious as possible, and we have endeavored to remove all considerations from these, except those pertinent to the discussion. We consider it extremely likely that the phenomena being considered here, are also artifacts in other types of artificial intelligence programs.

Analysis of Evidence

When branches in a tree search must be terminated prior to a legal termination point (according to the rules of the game), it is necessary to assign a value (an interim value other than win, lose, or draw) to the terminal node, which then allows comparison with other terminal nodes. This is usually done by invoking a static evaluation function. In games where a search to legal termination is not possible, no other recourse appears possible. An interesting phenomenon arises from the interaction of the artificial termination of the search and the fact that all the terms in the static evaluation function are evaluated at this terminal point. The result of this combination is that for the game playing program, reality exists in terms of the output of the static evaluation function, and anything that is not detectable at evaluation time does not exist as far as

the program is concerned. This interesting fact is present in all tree searches in any chess program that we Know of, and causes interesting aberrations in program behavior.

The class of aberrations defined above, we call the Horizon Effect. Examples of the The Horizon Effect have been observed by several researchers [11,5,2,1] in game playing programs. However the complete phenomemon has never received a name in the literature nor has its causes and effects been properly cataloged. The regimen of insisting on evaluation at a prespecified point in the search causes the following effects which seem peculiar to human observers. When the Horizon Effect results in creating diversions which ineffectively delay an unavoidable consequence or make an unachievable one appear achievable, we call it an instance of the Negative Horizon Effect. This is the phenomenon previously reported in the literature. It can best be shown by a typical example.

In Figure I it is White's turn to play, and for the sake of this example let us suppose the search is to be limited to three ply (we realize that the search usually goes deeper, however it is relatively easy to construct examples at any given depth, and we are choosing our examples for their expository simplicity). What will happen in the above position is that the program will try to play 1. B-N3 and after P-B5, 2. Anything, it is time to do a static evaluation. This usually consists of a material quiescence analysis, together with a calculation of the other coefficients of the evaluation function. The material quiescence analysis could consist of trying all capture sequences and accepting the minimax value if it is an improvement for the side starting the sequence. Other quiescence procedures are also possible, but in essence they should yield the same value. Now at the end of the above 3-ply sequence, the program will come to the conclusion that it will lose the Bishop on N3, and will continue its search for something better. It will eventually come upon 1. P-K5 and recognize that if now PxB, then 2. PxN is good for White. Therefore it will consider as best for Black to play PxP, after which White plays 2. B-N3. Since we are now at maximum depth, this position *will* be evaluated using the standard procedure. The quiescence analysis will show that White has saved his Bishop since there is no sequence of captures which will win the Bishop. Alas, it is only after the next move that the program finds out that the pon-caoture threat of P-B5 has not been met by this diversion, and it then looks for other ways of parting with material worth less than a Bishop in order to postpone the inevitable day when the Bishop will finally be trapped and captured. In this case 2. RxB would no doubt be tried next since after NxB, 3. B-N3, "saving" the Bishop by giving up the Rook for the Black Bishop is preferred to losing it. We have seen programs indulge in this type of folly for five to six successive moves, resulting in going from a position in which they *are* well ahead to *one in* which they are hopelessly behind.

A clever device to prevent this behavior was invented by Greeenblatt and is also used by the Northwestern University group [5,1]. This consists essentially of extending a new principal variation another two ply, to see whether the reason it was considered superior, will continue to obtain. In the above example, this will result in finding that the

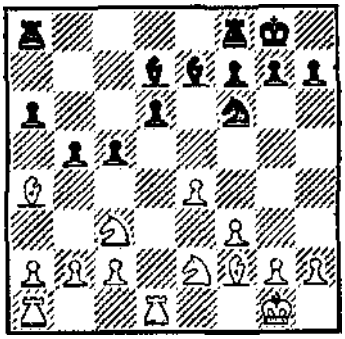

Figure 1
White to Play

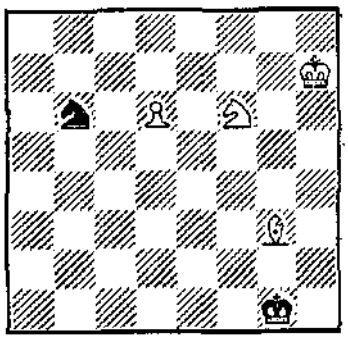

Figure II
White to Play

threat of P-B5 does not go away, and thus a potential sequence of blunders is averted. However extending a principal variation two ply can only discover whether a one move threat has or has not been dissapated. Threats requiring two or more moves can not be dealt with effectively in this manner. This is usually not noticed, since today's best programs perform at a level of skill where two move threats are rare and far from the major cause of, concern for their developers.

The Positive Horizon Effect is different in that instead of trying to avert some unavoidable consequences, the program grabs much too soon at a consequence that can be imposed on an opponent at leisure, frequently in a more effective form. This phenomenon has been largely overlooked in the literature, but is reported in a previous paper [2]. Figure II shows a flagrant example of the Positive Horizon Effect.

In this position it is White to play and the search is again to a depth of three ply. The program notices that it can play 1. P-Q7 and if Black does not now play NxP, 2. NxN, then it would get a new Queen. It sees that in this way it can increase it's material superiority. It may or may not notice that it will then have to face the formidable task of mating with a Bishop and Knight. The interesting thing about this position is that the maneuver I. B-K5 followed by 2. B-Q4 cannot be prevented and results in forcing the Pawn through to a Queen without letting Black give up the Knight for it, thus simplifying the win greatly. Here the important point is that there is a consequence on the horizon, and the program insists on realizing it within the horizon of the search as otherwise It, does not exist. As a result, a consequence which could have turned out to be very beneficial, turns out to only have a small benefit.

It is possible to find examples where wins are "thrown away" by such a premature cashing in. In fact, the main reason for the demonstrated lack of tolerance of complexity of today's chess programs is that their evaluation function insists on maximizing, in terms of a preconceived set of evaluation terms, anything that it detects within the search horizon, and thus all too frequently destroys an advantageous situation before it really has a chance to bloom.

An example of the Positive Horizon Effect illustrating a throwing away of a positional advantage is shown in Figure lll. Here, if the evaluation function is aware of the beneficial effect of controlling an open file, and if the search is again being conducted to three ply, the most likely continuation will be 1. PxP ch, PxP, 2. R-KR1 with control of the open file and "some advantage". The fact that on the next move Black can answer R-KR1, after which White's advantage has largely evaporated is not recognized. Neither is the Key fact that Black can do absolutely nothing to prevent White from opening the file whenever he likes (for human players there is the dictum "do not open a file until you are ready to use it"). However today's programs would almost certainly reject the correct 1. R-KR1 since after Black plays R-KR1 and White plays 2. PxPch, it is time to invoke the quiescence procedure which produces PxP. Now in contrast to the earlier variation, White does not control the open file. Nor would he if any other 2nd move were played. Clearly, a program could recognize the value of playing 1. R-KR] before J. PxP ch, only if it were secure in the knowledge that the file can be Opened at a later time by PxP and that if Black plays PxP, he will merely incur an equally difficult problem in defence of the KRP as he has now in defence of the KR-file. In fact having once played i. R-KR1 and getting the answer R-KR1, a program that has reasoned thus far should have little difficulty in now playing 2. R-R2 since opening the file at the present moment is not advantageous and making room for the other Rook could help. It should be noted that incorporating the human players' dictum appears extremely difficult as the issue of "ready to use it" is one requiring dynamic judgements, in which even good human players make mistakes at times. However, we feel that the construction of lemmas as explained in a later section, will go a long way toward providing better evaluation mechanisms at the search horizon.

Another basic problem, the need for a global strategy, is shown in Figure !V. Here everyone of today's programs would conduct a 5-ply search and then play 1. K-K3. A summary of its findings during this tree search might run a follows: it decided that P-B7 would lose the pawn to K-K2, and therefore decided to move the King to the most 'central location available (this is a quantity recognized by the evaluation function). On the next move, having already achieved its "optimum" position, the program would be faced with a problem that all hill climbers face when they reach the top*. How to back down as little as possible? Accordingly there would occur either K-Q2 or K-B3. The point Of this whole example is to show the hopeless hill-climbing characteristics of the present program design. In the given position, even a poor human player would recognize that there is nothing to be gained by the above maneuvers. The real problem is that today's programs mix their strategical and tactical objectives during the search. Thus the above position could be handled effectively if a tactical

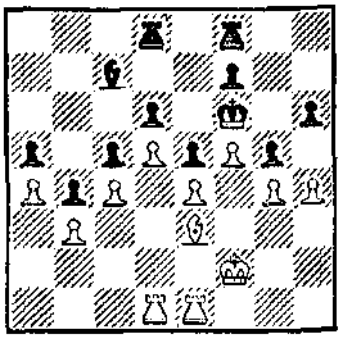

Figure lll
White to Play

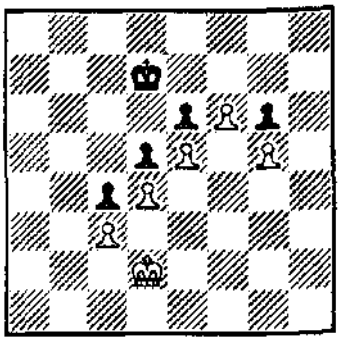

Figure IV
White to Play

search were first done and this came to the conclusion that P-B7 only resulted in losing the Pawn. There being no other tactical tries, control would then revert to a strategical module which would try to improve the position of any and all pieces. Since, in this simplified situation, we only have the King as a candidate, the next step would be to try to find an optimum or near optimum position for the King and determine if it could get there. Here we must not rely solely on a static, preconceived notion of centrality, although that certainly is a part of the picture, but more importantly we seek a functional optimum. This can be found by noting that the Black KNP and KP are not defended by Pawns and could possibly be attacked by the King, and also that our own KBP could possibly benefit from having our King near it. Next, a null move analysis could be carried out, consisting of moving the White King around without looking at intervening moves, to see if we can find the optimum path to any of the desired squares. This will then eventually yield the correct idea of infiltrating with the White King via QR3, which wins easily. Admittedly the control structure that could evoke such behavior would present some problems. Most of the problems in chess are tactical (immediate) problems and for this reason, the lack of global ideas is frequently obscured in today's programs. However, it is absolutely necessary to be able to generate global goals in order to avoid hill climbing behavior.

We have above touched only on the relatively simple problem of finding the correct way to proceed. A far more difficult problem, which would also have to be faced by the Master strength program, is to judge whether the position can be won or is a draw. A simple "pawn ahead" judgement is not enough. There may be other endgames from which to choose, in which

the program is also a pawn ahead. In the position being discussed, for instance, if a further White Pawn were at QN4, and a Black Pawn at its QN4, the position would be *a* draw. Clearly *dynamic judgements of this* type are absolutely necessary.

In Figure V, we see a much better understood problem than any of the above. It is the problem of calculating in depth. Here White can execute a mating combination requiring an initial Queen sacrifice and nine further moves, a total o(19 ply as follows: 1. Q-R5ch, NxQ, 2. PxPch, K-N3, 3. B-B2ch, K-N4, 4. R-B5ch, K-N3, 5. R-B6ch, K-N4, 6. R-N6ch, K-R5, 7. R-K4ch, N-B5, 8. RxNch, K-R4, 9. P-N3, Any, 10. R-R4 mate. This combination was played by a former World Champion while playing a total of 20 games simultaneously. The reason no program that looks at 10 to 20 alternatives at *every node can play* the correct move is that the principal variation to justify the initial queen sacrifice extends much, much further than the 5-ply depth that is about all that is possible with a program that gets buried in the exponential explosion of investigating 10 sprouts from *every* node. Now it is quite possible to play Master level chess without playing such long combinations. However, in the author's experience one must at least once a game be able to *took 14* or more ply ahead. As for as the above example goes, we believe that 992 of all Masters would solve it as well as a high percentage of Experts and Class A players. What is really difficult about the example is not the *simple* unravelling of the main line, which having few branches is fairly linear, but the conception of the position, and that such a solution involving chasing the King up the board might exist in it.

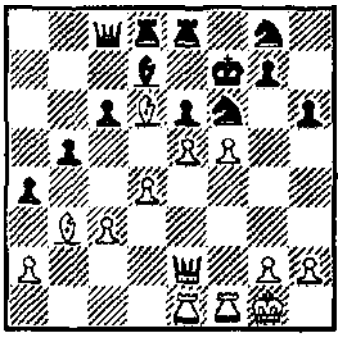

Figure V
White to Play

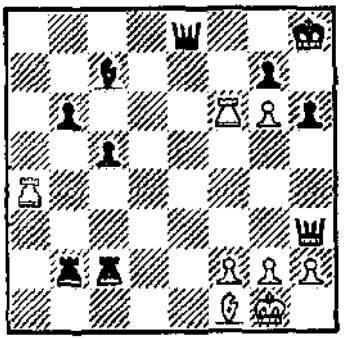

Figure VI
White to Play

One could argue that just because good players can solve such problems, this does not show the requirement for the program to see to such depths in order to play at the Master level. What this would mean is that the program would have to rely almost exclusively on static, non-tree-search computations for its moves. But we have already shown in examples II and III that static notions must be combined with dynamic tests in depth in order to yield correct results. So a program that could not look 10 ply ahead would be subject to any five move threat that comes along. Even though the main thrust of most such threats could no doubt be muted, it would be inevitable that some concession would have to be made. This type of thrust and parry is at the heart of Master play. Even more importantly, a program that cannot look 10 ply ahead could never conceive a five move threat of its own which is dependent on adverse action. The evidence is quite overwhelming.

Another interesting phenomenon, that of reality or illusion, that afflicts all of today's best programs can be seen in Figure VI. Here it is White's turn to play. The first thing that the evaluation function will discover is that White has both of his Rooks "en prise" {capturable by the opponent under favorable conditions). If this position has occured at some node which is eligible for sprouting, then moves that move either of the rooks to a "safer" place will receive good recommendations. If the node is a terminal node, then it will be considered as not satisfactory for White, as it is presumed that at least one of the Rooks will be lost. In actuality, neither of the Rooks is in danger. If BlacK plays QxR then R-B8 mate, and if PxR, then QxPch, K-NI, Q-R7ch, K-B1, P-N7ch followed by P-N8-Qch wins quickly- Even stranger is the fact that if this position occurs somewhere in the tree below the top node, and if, say, two ply earlier White had played RxP(KB6) as a sacrifice which it turns out could not have been accepted, then in today's programs there would now be no knowledge of the sacrifice at KB6 when the position is tendered for evaluation two ply later. Rather the Rook would be considered en prise. Indirect defences of this type are seen all the time in Master chess. *Clearly,* if a program *aspires to this level it must* be able to handle such problems. Part of the solution consists in noting the functional overloading of the pieces that are thought of as doing the • capturing. Here the Black Queen is guarding a check on the back rank apart from attacking the White Rook. Also the Black KNP is guarding a Pawn and a check, while attacking, the White Rook. However this is not enough, since it is quite possible that the checks that are being defended against are quite harmless, and it would be folly to try to determine, without further searching, the exact potency of every check on the board.

Another problem, that of dynamic evaluation of material, is depicted in Figure VII. Here with either side to play, White's pawn cannot be stopped from queening, while Black's pawns are going nowhere fast. Yet there is no doubt that every one of today's programs, if playing Black would refuse a draw in this position, and it is also very clear that only a *very* weak human player would offer a draw with White. The programs' rationale is that three passed pawns are better than one. The problem here is one of recognizing the dynamic potential of the White passed pawn which cannot be caught. It is true that in this case the job can be done statically be merely noting the distances of the White Pawn and the Black King from the queening square, however, if the Black Pawns

were ell advanced three squares, the computation would have to be done dynamically, since there is a possibility they may arrive first. Similar dynamic ideas, which no program can at present handle well, ere the notion of a defenceless King by reason of no surrounding men of his own to help defend him, and the notion of cooperation among various men rather than only assessing the goodness of their individual positions. Such notions require dynamic exploration to determine the degree of their applicability in a given position. However, in a program where terminal evaluation must be done very quickly because of the large number of nodes that must be evaluated, such luxuries are not possible. We are here directly confronted with a basic limitation of the generate and test approach, when it does not allow enough time to do a detailed evaluation of the nodes visited.

Our last two examples deal with situations that present-day programs can handle. However, the method by which they do this is terribly inefficient and could not be used if one wanted to do tree searches which could extend even a little deeper than the current five ply. The first of these problems is the problem of defence. It is relatively easy to recognize attacks and develop criteria for judging the value of most attacks. However, this is not so with defence. The problem is that in order for a defence to exist, a threat must first be known. All threats are not of the simple type such as threatening a capture, and it is precisely this other type of threat, which shows up in the backed up value of the current variation, that is not easy to counter because we only know the magnitude of its effect. Figure VIII shows a position of this type. Here it is Black's turn to play and the search is being conducted to a depth of five ply. If Black plays a normal aggressive move such as 1.— P-R7, he will find that after 2. Q-K8 ch, RxQ, 3. RxR he is mated. The search will then eventually revert to the point where Black played P-R7. Now in most of today's programs we would be armed with the killer heuristic (which says that against any new proposed move try the "killer" Q-K8ch first). This would indeed result in the efficient dismissal of the next 15 or so moves likely to be tested. However the fact remains that each of these alternatives is being served up in a generate and test mode, and the program can consider itself fortunate if it discovers the only defence (Q-K5) before it has exhausted half the legal moves.

Our final example in Figure IX shows ' another subtle consideration. In this position, programs that look five ply deep have an excellent chance of finding the mate in three moves; 1. BxPch, K-R1, 2. B-N6ch, K-NI, 3, Q-R7 mate. If such a program, due to the fact that White is behind in material, were only to look at captures of pieces of greater or equal value to the current deficit, and checks (an assumption which requires some preprocessing) and to stop at five ply depth (for which it would be difficult to establish a logical reason), there would still be about 100 bottom positions to examine before the mate is found. Here any tournament caliber human player would recognize the situation immediately as one of a set of Queen and Bishop mates. He would only have to determine the functional need to guard the King escape square at KB2, to determine what the correct sequence is and that it does lead to mate. The critical thing here is not that a program couldn't find the mate once the diagrammed position is reached, but that in advanced chess play such situations occur frequently

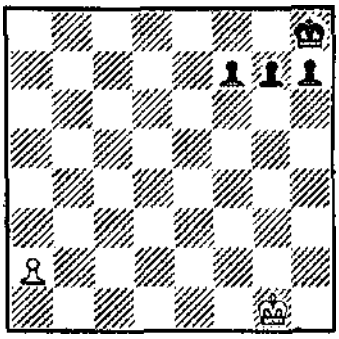

Figure VII

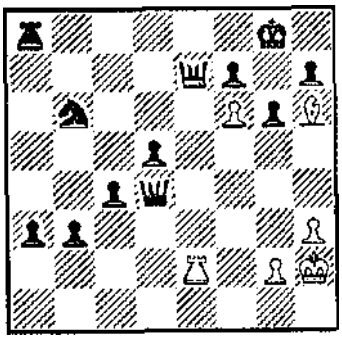

Figure VIII
Black to Play

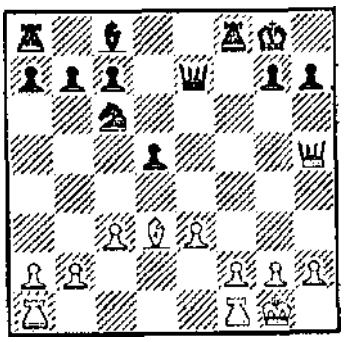

Figure IX
White to Play

in deep parts of a tree as a reason why some other move fads. If a program spends 100 nodes investigating such a well known pattern, then there is a definite limit on the amount of work the program can be expected to do. The answer here quite obviously is to have a repetoire of frequently occurring patterns available to the program together with some guidance to determine the exact applicability of any particular pattern. In the above case, recognition of the Queen and Bishop functionally bearing on the undefended KR7 square, together with the position of the Black King hemmed in by some of its own pieces is the basic pattern. The dynamic analysis reveals that the King could escape over KB2 if this were not kept under continued guard. With these constraints, the number of variations to be examined are very few.

gome Conclusions

Let us examine some potential models of computer chess. All the complete models are clearly too time

or space consuming. Therefore, the most reasonable course appears to be to rely upon models that construct • trees of possibilities but with some limitations imposed upon the growth of the trees. Now depending upon how we define these limits, we have *a* tractable problem. The real question, and that addressed by this paper, is how these limits can be defined and implemented in order to include the range of performance exhibited by chess masters while still keeping the problem tractable,

Let us summarize the requirements noted earlier:

1) In examples I-III we have seen the Horizon Effect in operation. We have also seen that the two-ply extension of each *new principal* variation is only a stop-gap measure, which prevents one move debacles (anyone who doesn't believe this is invited to try Figure XI out on his program). What can be done about the Horizon Effect? Clearly the problem is due to the fact that some term in the static evaluation function is evaluated "prematurely". Prematurely here means that a noticable' change in the value ot the term can be forced, without any compensatory change in any other term(s). From this, one can deduce that there can be no arbitrary depth limit imposed on the search. The decision as to whether to terminate the search at a node or continue, has to be a function of the information that exists at that node and how this relates to the quiescence of each and every term in the evaluation function. For instance, if we have *an* evaluation function that would consider it bad to have a bishop blocked in by its own pawns, then some effort must be expended to determine the permanency of such situations. In general what is required is a procedure to determine the quiescence of every term in the evaluation function and in cases of non-quiescence, a procedure for generating moves or applying some static means of reaching a quiescence decision. This should not be construed as meaning that perfect knowledge of the future status of each parameter in the evaluation function is required, in fact some practical maximum depth or time limits must exist. Thus only a finite set of resourses can be expended to determine the true future status, and some controlled error will no doubt neve to be tolerated. However, the resulting error by this method should be orders of magnitude smaller (a -so-called judgement error) than the errors produced currently by the Horizon Effect. In present day programs, quiescence is pursued only for the material parameter. And even this frequently does not work out satisfactorily, since usually only captures are considered, while forks, mate threats, etc. are ignored.

2) From example IV we see the need for having global goals and being able to determine something about the feasibility of such proposed goals. This may involve procedures of considerable complexity in order to answer basic questions about the value of any node. All of which adds to the potential evaluation time required at a node.

3) From example V, the program must on occasion be able to calculate precise variations to a depth of 14-*ply* and possibly *more.* This *in no* way means that every move should be calculated to this depth nor that when a move is, that every branch would go to this depth also. However, the basic facility to allow probing to at least this depth must exist.

4) From example VI we see the need to diagnose certain dynamic properties of positions, and the requirement to communicate such data to other nodes in the tree. This need exists in order to avoid faulty interpretation and the necessity of otherwise "discovering America" over and over again.

When conditions are detected that make a statically calculated decision incorrect (in this case that a rook is en prise), the variation that discovers this fact must during tree back-up assemble the conditions which are necessary for this contradiction to remain true, This new truth should then be stored as a lemma applicable to all nodes in the tree below any node for which it is discovered to be true. The lemma states the conditions under which it remains true, and at each succeeding node it is assumed true unless something that counters one or more of the conditions of the lemma has occurred. In that case the validity of the lemma would Have to be re-examined. It is important to express the lemma in a language which is neither too detailed nor too fuzzy. The former case would result in continuous re-examination as changes in single conditions, which by themselves do not upset the validity of the lemma, have to be looked into. The latter case *could* result in being unable to define or detect a critical change in the truth value of the lemma. It should of course be apparent that lemmas can concern themselves with other things than the material issues presented in the example. For instance, it should be possible to posit a lemma about the conditions under which control of *an* open file is retained.

It is interesting to note that a key to detecting that something may not be as it appears statically, is the use of a functional analysis. In example VI, the initial indication that neither of the Rooks is capturable is that each of their attackers is also defending something of importance. Sometimes it is possible to resolve such function conflicts statically by noting that another piece can assume the required functional role without itself becoming overburdened. When this is not possible, the validity of a potential function conflict must be established dynamically by tree searching. A general discussion of the use of functional relations in chess perception and reasoning can be found in Newell and Simon [7]. A good discussion of paths and their obstruction can be found in Botwinnik[4].

5) From the defence problem in example VIII, we see a need for some similar communication within a search tree. A proper description of a set of undesirable consequences can save tremendous effort in finding problem solutions if such exist, or moving on to more fruitful endeavors if not. Again, the adequacy of the language is important as it must be used to test whether the set of consequences were caused by the latest move, and to provide an input to move generators

that could find an appropriate answer to the problem. For this purpose, functional relations which describe attacks that occured, and path information which describes paths traversed by moving pieces and paths over which threats Occurred, appear to be required elements of the language.

6) The functional relations mentioned in the previous examples are in a sense patterns involving two pieces or a piece and a square. Certain clues can be gained by searching these patterns when they focus about a common square or piece. However, from example IX we can see the need for a still higher level of pattern abstraction. Here we are looking for groups of pieces which form a pattern around some interesting focus. In the example cited, the KR7 square with the White Queen and Bishop attacking it, and the Black King are the focal points which should suffice to index into the correct pattern, which will then produce a pointer to a routine for deciding if we are confronted with an exploitable instance of the pattern in question.

Above, we have assembled the beginning of a set of requirements for a program that could have the power to play Master level chess. It does not take long to dismiss the possibility of extending the current generation of chess program to meet the above requirements. It is quite enough to realize that such a program requires about a factor of 20 of additional time for each additional two ply of depth that it searches.

In 1958 Newell, Simon, and Shaw [8], argued that "As analysis deepens, greater computing effort per position soon pays tor itself, since it slows the growth in number of positions to be considered". This is well substantiated in the ACM tournaments which have convincingly shown the superiority of programs that search a subset of legal moves and evaluate a moderate amount, over programs that search all legal moves and evaluate little. Clearly it is time to move again, and more substantially in the direction of more evaluation and less search. The requirements demonstrated in this paper show a need to do possibly ten or more times as much processing at a node than is currently done. This means that, for equivalent computing power, we are faced with generating trees of at most 5000 nodes distributed throughout the search space. The Greenblatt and Northwestern University programs have an effective branching factor (where number of bottom nodes - $BF^{Kr,H}$) in excess of 6. If it is assumed that the search is limited to 14-ply, then the branching factor must be less than 1.9, if we are to stay within 5000 nodes.

Actually this is a meaningful measure Only for trees which have a maximum depth. In order for a tree of no maximum depth to converge, a necessary and sufficient condition is that for any arbitrary node 5 < i P[i]) < 1 (where P[i] is the probability of i sprouts). Clearly the less *I* (i P[ij) is, the more rapidly the tree will converge. One can achieve such a decrease, with increasing depth of the tree, by being able to compare the state at the present node with the states of earlier nodes in the tree branch being investigated. Comparisons involve how earlier expectations are holding up, and whether moves that are eligible for testing have appropriate thematic relationships to what has gone before. The number of such comparisons grows linearly with depth thus providing ever more conditions for stopping the search or not Investigating an arbitrary move.

To guide the search we need mechanisms which can at linear cost provide analysis at a node so that the exponential cost of discovery and/or verification due to tree searching Is drastically reduced. It appears reasonable that the more powerful (in the sense of greater depth) the prediction mechanism, the better the effect on program performance. Here the functional analysis and pattern recognition mentioned earlier clearly are destined to play a part, with the former being an essential element of the latter. Also the communication of defensive requirements appears vitally necessary. In fact since dissatisfaction with a result is a relative matter, one could consider using backed-up descriptions to discover ways of heightening the success of whatever is being attempted at present.

Lastly, one can see the overriding importance of quiescence of concepts being used in the evaluation procedure. The evidence is quite overwhelming that the attempt to drive all evaluations into a quiescent state should be the major force that determines the shape of the tree. Thus, while today's programs use up nearly all their time trying to assure tactical quiescence, this will now have to be done by less complete methods in order to make way for the additional facilities required. It is interesting to compare this derived role of quiescence as the main guiding force, with the control structure of the 1958 Newell, Simon and Shaw program [8] which was apparently derived from a concern with human behavior.

The conditions and the model we have set forth appear to be necessary for Master chess. However, they are almost certainly not sufficient. Masters Know a great deal of chess knowledge which has as yet not been encoded in any program, and would probably have to be placed in long-term memory for occasional reference, We have avoided discussing what a minimum quantity of such data might look like, since until the necessary mechanisms for its use are in place, so that it would be possible to do some experimentation, there would be little scientific validity in such speculation. There is also the problem of doing at least some learning in order to avoid repeating obvious errors in identical situations. However, an organization ' which takes account of the conditions noted here is almost certainly necessary to make significant progress beyond the present state of the art, and the model appears extendable to the problems of learning and further pattern encodings, as these prove necessary. In the immediate future, the major problem appears to be how to produce a search of the economy of that proposed while retaining at least the same reliability as evidenced by today's programs which use a more complete search strategy.

For those who feel that our simple examples would not challenge their program design, we include Figures X-XII which are moderately more difficult. In Figure X it is White to play and win. In Figure XI it Is Black to play and not do something foolish; and we are not thinking of the obviously foolish 1. QxR, but of other foolishness derived from this by the Horizon Effect. In Figure XII it is White to play and win. We expect that a human Class B player would have no difficulty with any of these.

Progress To Date

Since July 1972 the author has had a program, whose general objectives are those outlined above, running at Carnegie-Mellon University. In this section we wish to report the progress that has been made. As yet the program only concerns itself with tactical {material) issues; however, it is felt that the techniques being developed are general and applicable to the other dimensions of *chess* evaluation. The program notes, among others, the relations of attack, defence, blocking functions and paths along which such activities occur. Thus, it has a language which is considerably more powerful than the notions Of a legal move, which is the analytical element of most of today's programs. Each chessboard is described in this language. This characterization greatly facilitates finding good "sacrificial" moves and moves which disturb the defence relations among pieces, while not hindering the evaluation of other types of moves. The program has an expectation level associated with *every* search it conducts, and there are mechanisms for raising and lowering this when results outside a range of acceptability from the expectation can be forced. There is a general causality facility which is used both for defence and improving attacking ideas. This facility *can* always detect whether a given set of consequences were not caused by the last move tried and thus constitute a problem inherited from higher in the tree. It does this by comparing a description of the consequences with a description of the move tried. However, it is not nearly as adept at detecting causality relations as an experienced chess player. Still, the facility never deduces causality when causality does not exist, end deduces non-causality about 607. of the time that this is appropriate, putting it well ahead of contemporary programs. No dynamic re-evaluations or lemmas have been implemented as yet, but these are planned for the *near* future. Higher level patterns are planned for a stilt later stage.

The program is at present able to investigate an average position for tactical quiescence to a depth of 9-ply, generating from 100 to 2000 nodes. The tactical reliability of such searches is somewhat better than that of the average program that participates in the annual ACM tournaments. Most of our effort in tree searching has been in an attempt to improve the likely correctness of a proposed *move.* This is achieved not only through the representation we use, but also through some 15 stopping, pruning and reordering rules. Thus the effective branching factors of our trees vary between 1.5 for positions where a clear-cut result is obtainable, and 3.0 in positions where there is tactical complexity, but no way of gaining an advantage exists. Although this compares very favorably with the effective branching factors of about 6 found in today's programs, we consider branching factors of 3 to be still much too high for our purposes. Thus, we occasionally grow a tree of 20,000 nodes because the program gets side-tracked in *an* inconsequential part of the tree. However, there are still several tree control algorithms we plan to implement, which should prevent such behavior by selective jumping around in the tree. We expect In the not too distant future to extend the depth limit of the program significantly beyond its present 9-ply. Even at its present depth, it is less affected by the Horizon Effect than programs with less

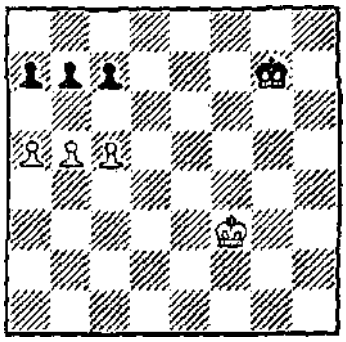

Figure X
White to Play

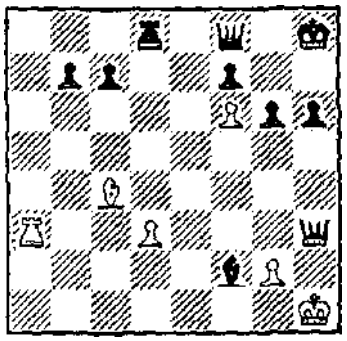

Figure XI
Black to Play

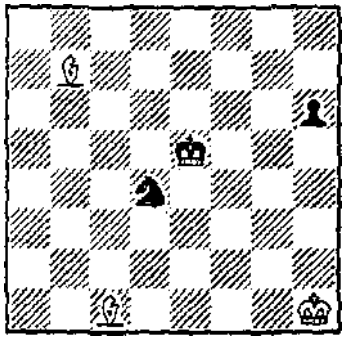

Figure XII
White to Play

deep searches. For instance, it has no trouble at all in Figure XI, deciding that there are no diversions which can lead to an ultimate capture of the White Rook.

When the program is confronted with the defensive problem of Figure VIII, after the initial variation starting with 1.— P-R7 is tried, a description Of what happened is backed up along with the value of the position at branch termination. At each successive node during backup, if the result is unsatisfactory, the backed up description is compared with the description of the move made. This is done to see if what happened could have been caused by the latest move tried at that node. If not, a defensive move generator then generates the moves that *can* do something about the description. The description contains such information as the names of the pieces involved in the action, the path (if any) that they traversed, targets attacked when they arrived at their

destinations, etc. If no defensive moves are generated and if no worthwhile aggressive moves remain to be investigated, it is assumed that the problem cannot be solved at this node of the tree. When the search reverts to the point where 1.— P-R7 was tried initially, an examination of the description reveals that the threat could possibly be countered by moving the King, guarding the K8 square, blocking the path of the Rook from K2 to K8, capturing the Rook or the Queen, or blocking or getting ready to block the KB8 square across which the check passes. From this description six moves *are* generated: K-RI, *N-Q2,* R-KB1, Q-K5, Q-K4ch, and Q-K6. It then does not take the program very long to determine that Q-K5 is the only defence (which in fact leaves Black in a winning position).

The program has been tested extensively on positions which one finds in chess books which teach how to play good tactical chess. In such positions a more Or less "hidden" series of moves allows the player on move to achieve a decisive advantage. In working on the first 200 examples in thr book "Win at Chess" [9], the program solves about 607- of the middle game problems in CPU times ranging from a fraction of a second to eight minutes, with an average of about 80 seconds. This *is* despite the fact that the principal variation sometimes extends to nine ply. Occasionally the program finds forced mates in the absolute minimum number of nodes. It also has found improvements on the analysis contained in the book, which the experimenter himself did not notice. The main reason that the program is not able to solve 100% of these problems is that several functional relations a*nd many* analytical routines are not yet implemented.

On the basis of tests to date, we find that the language of functions, paths, etc. is very useful for passing messages of the kind described above. We expect that it will also serve for the characterization of thematic relationships which would mitigate against a *move* being tried when it is "out of context" with previous moves. However, there seems to be little doubt that more powerful languages are possible. These may be necessary to effectively implement the lemma ideas, since the function language may be so detailed that investigating lemmas expressed in it will prove uneconomical. Further, we conjecture that continuing progress in chess will be dependent on the invention of ever higher level languages in which chess concepts can be expressed. Each such language level would then have an assymptote, defined by the power of the language, beyond which it would not be possible to improve the strength of the program, given that only a certain amount of time was available to compute a move. Also as concepts are agglomerated into ever higher level concepts, we expect that they would get to be more fuzzy and would require a more complex control structure than used a present in order to produce the same level of reliability as can be obtained with less fuzzy concepts. We feel that this increased conceptualization is evident in the history of chess, and should make it possible ultimately to equal and exceed the performance of the best human players.

Acknowledge ment

The author wishes to acknowledge the roles of Professors A. Newell and H. A. Simon in stimulating his interest in functional analyses of chess positions. The work reported herein was supervised by Dr. Newell.

References

1. Atkin, L.R., et. al., "Chess 3.0 - An Experiment in Heuristic Programming", Unpublished. 1971.

2. *Berliner, H.J*, "Experiences Gained in Constructing and Testing a Chess Program", Proceedings of the IEEE Sysmposium on Systems Science and Cybernetics, October 1970.

3. Bernstein, A., et. al., "A Chess Playing Program for the IBM 704", Proceedings of the Western Joint Computer Conference, A1EE, March 1959, pp. 157-159.

4. Botwinnik, M.M., Computers, Chess and long-range Planning, Springer Verlag, 1970.

5. Greenblatt, R.D., et. al., "The Greenblatt Chess Program", Proceedings of the 1967 Fall Joint Computer Conference, pp. 801-810.

6. Kotok, A., "A Chess Playing Program for the IBM 7090", Bachelors Thesis, MIT 1962.

7. Newell, A. and Simon, HA., Human Problem Solving, Prentice-Hall, 1972

8. Newell, A., et. al., "Chess Playing Programs and the Problem of Complexity", in Computers and Thought, E.A. Feigenbaum and J. Feldman (Eds.), McGraw-Hill, 1963.

9. Reinfeld, F., Win at Chess, Dover Books, 1958.

10. Shannon, C.E., "Programming a Computer to Play Chess", Philosophy Magazine, Ser. 7, Vol. 41, No. 314, March 1950, pp. 256-275.

11. Strachey, C.S., "Logical or Non-Mathematical Programs", Proceedings ACM National Meeting, 1952, pp. 46-49.