THE CHESS MACHINE: AN EXAMPLE OF DEALING WITH A COMPLEX TASK BY ADAPTATION

Allen Newell

P-620

28 December 1954

Presented before the Western Joint Computer Conference, March 1, 1955 at Los [illegible]eles

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA

The modern general purpose computer can be characterized as the embodiment of a three point philosophy:

1. There shall exist a way of computing anything computable;
2. The computer shall be so fast that it does not matter how complicated the way is;
3. Man shall be so intelligent that he will be able to discern the way and instruct the computer.

Sufficient experience with the large machines has accumulated to reveal the peculiar difficulties associated with these points. There has been a growing concern over problems which violate them, and instead satisfy:

1. The relevent information is inexhaustible;
2. The set of potential solutions is neither enumerable nor simply representable;
3. The processing to be accomplished is unknown until other processing is done;
4. An acceptable solution is required within a limited time.

Most design problems, including programming a computer, are of this nature; so are the very complex information processing tasks like translating languages or abstracting scientific articles. The current arguments about thinking machines and general purpose robots also revolve about whether computers can deal with problems of this general nature.

The problem of playing good chess certainly falls into this class of ultracomplicated problems. It is a useful type case for general discussion because the nature of the task and the complexities surrounding it are common knowledge. Further, it already has something of a history (1,2,3,4).

The aim of this effort, then, is to program a current computer to learn

---

(1) Numbers in parentheses refer to similarly numbered references at end of paper.

to play good chess. This is the means to understanding more about the kinds of computers, mechanisms, and programs that are necessary to handle ultra-complicated problems. The limitation to current computers provides the constant reminder that the heart of the problem lies in the limitation of resources, both memory and time. This aim would be somewhat ambitious even if all the details were available. The paper will actually be limited to presenting an overall schemat which appears feasible, and relating it to some of the critical problems which must be solved. The reference to learning expresses the conviction that the only way a machine will play good chess is to learn how. Although learning considerations have been prominent in the thinking and motivation behind the machine, attention will have to be restricted to the performance system; that is, to those features which are necessary in order to play the game. However, some of the learning potentialities implicit in the performance system will be discussed.

The work presented here represents the early phases of an attempt to actually construct such a program for the Johnniac, one of RAND's high speed computers.

Before starting it is desirable to give some additional conditions of the problem. From now on the computer with program will be called the "machine", and the various parts of what it does will be called "mechanisms". The problem is not to construct a machine which can induct the rules of chess by playing; it will be instructed concerning the legalities. Finally, the machine is only to do the job of one man; it will require an outside opponent, human or otherwise.

Problems

As everyone knows (5), it is possible in principle to determine the

optimal action for a given chess position. First compute out all continuations to the bitter end. See whether they win, lose, or draw; and then work backwards on the assumption that the opponent will always do what is best for him and you will do what is best for you.

The difficulty, of course, is that this "in-principle" solution requires a rather unnerving amount of computing power, (1) and doesn't give any clues about what to do if you don't have it. It will provide us, however, with a short checklist of problems which must be solved if the computing requirements are ever to shrink to a reasonable size.

The most striking feature of the "in-principle" solution is the tremendous number of continuations. This is accounted for by both the number of new consequences that appear at each additional move into the future, and the large number of moves required to explore to the end. This provides two problems:

1. The consequences problem, or which of the possibilities that follow from a given proposed action should be examined;
2. The horizon problem, or how far ahead to explore.

The possibility that one might stop looking at some intermediate position, only raises a third problem:

3. The evaluation problem, or how to recognize a good position when you see one.

Another feature of the "in-principle" solution is the identical examination of all the possible alternative actions. Despite the similarity in describing both present and future moves, it is worthwhile keeping distinct the actions that are actually available at a move, and from which a choice must be made; and the future consequences of these actions, which may include,

among other things, limitations on the alternatives available in the future. Hence, we have:

4. The alternatives problem, or which actions are worth considering.

These four problems; consequences, horizon, evaluation, and alternatives; will be sufficient to keep us aware of the difficulties as we search for a set of mechanisms to play chess. Solutions must be found to all of them if the machine is to play good chess with reasonable resources.

Overview

There is a common pattern to the solutions to be described here. In all of them the machine uses very partial and approximate methods. It is as if the machine consisted of a vast collection of rules of thumb. Each rule is a much oversimplified expression of how the machine should behave with respect to some particular aspect of the problem. The rules are of all kinds: chess principles to follow, measurements to make, what to do next, how to interpret rules of thumb, and so on. These are so organized that they form the necessary qualifications and additional specifications for each other. Each rule is essentially a machine program. At any particular instant the machine is under the control of some such rule, or shunting between rules under the control of a master program.

The main effort of the paper is devoted to describing how such a set of rules can be defined, and organized to acheive solutions to the four problems; and thus provide a schemat for a machine which puts all these pieces together to play chess. Only minor effort is devoted to indicating the detailed structure of these programs at the level of machine code.

One aspect of the underlying coding does require attention, and is dealt with at the end of the paper. The large number of rules, their complexity,

and the necessity for adding new ones and modifying old ones, implies the use of a fairly extensive general purpose language. That is, all these rules are to be given in this language or pseudo code as it might also be called. Hence, each use of a rule must be preceded by an interpretive step. However, a few programs suffice for using any and all of the rules that might be required in the machine.

Preliminaries

Let us start by providing the machine with a few basic facilities. Each chessman and each square of the chessboard needs a name, suitably coded into binary bits. A fixed set of addresses are set aside to hold the current position. This can be given as a list of the men with the squares they occupy, including a "zero" square if the man is off the board. The machine can accept an opponent's action by reading a punched card. This can be given as a list of the men which have been moved, along with their new locations. Thus an action involving a capture has two terms: one giving the new location of the man that captured, and a second giving the location of the captured man as the zero square, that is, off the board. The machine obtains the new position by substituting in the old one, which is already stored. It can also output its own action on a punched card.

The machine must also be equipped to answer an array of elementary questions that recur constantly, such as, "can a given man move to a given square?" or, "what man is blocking the Queen Pawn?" Each of these questions can be answered by a straight forward and not-too-lengthy investigation of the current position. The number of actual programs needed is within reasonable bounds since the more complicated questions are combinations of the more elementary ones.

Assume, then, that the machine has these basic capabilities. They have solved none of the four problems.

Goals and Tactics

Suppose, in the midst of a game, the machine (which is White) has stored in a suitable place the following expression:

att (BKB, WKR).

The machine interprets this as: "Attack the Black King-Bishop with the White King-Rook." Attack will be considered to mean a successful attack, which in turn means that the attacking man (here, the Rook) is capable of capturing the object of the attack (here, the Bishop) with a relative gain in material after the smoke of engagement has cleared. Thus, for a given position, the attack is either successful or not; and the machine can determine this by a sequence of those elementary programs given it earlier. Call such an expression a goal, which is either achieved or not achieved for any given position.

Now, suppose the machine were to start tracing out continuations. It could determine at each new position it arrived at whether or not the goal was achieved. If it is, the machine could stop searching. This might provide a solution to the horizon problem: have a goal and only explore continuations until a position is reached where the goal is achieved.

The machine can represent such a set of continuations as a branching net, or tree, of actions. Each action is linked by some kind of indexing to the immediately preceding action, and to each action is associated by this indexing a number of other actions that might possibly follow. Such a tree is a tactic for a given goal if it always terminates in positions that achieve that goal. Further, a tactic will always indicate a single action for the

machine to take, and many actions for the opponent.

But if the machine has a goal like "attack the Black King," it is right back with the original difficulty of searching for continuations which end in checkmate. This merely indicates that in general one can hardly expect to find a tactic for a given goal. But on the other hand, one sometimes will: men do get captured and games do get won.

By some means let the machine acquire in memory a second goal, which is indexed to the first:

blk (BQ,BKB) → att (BKB, WKR).

The machine interprets this as: "Block the Black Queen's ability to recapture where the Black King-Bishop is." If this ability to recapture was in fact one of the deterrents to taking the Bishop with the Rook, then achievement of this second goal could be considered as an intermediate aid to achievement of the attack. Such a goal will be called a subgoal. It may itself have subgoals, and thus the machine may acquire rather large networks of goals.

Now, instead of trying to construct a tactic for the attack goal, the machine can search for a tactic for the subgoal of blocking. Perhaps it will find one. If it cannot, then it needs either other subgoals for the attack or some subgoals for the block.

The single large search for winning positions has been replaced by two searches, each terminating the other. The machine searches for intermediate goals; it stops this search when it finds subgoals it can achieve. It determines the question of achievement by a search for particular sequences of action. This search terminates with a completed tactic, that is, when the sequences end in goal achievement.

But when to stop if the tactic search continues to be unsuccessful?

The mechanism of subgoal formation operates only if the signal has been given to terminate further search for a tactic. Before discussing this, it is appropriate to consider another aspect of the total problem, which, after a twist or two, will lead back to this same decision problem.

Likelihoods

The mechanisms introduced so far have only yielded a solution to the horizon problem. For instance, they do not provide a solution for the consequences problem. It would still seem the machine would examine all branches at any given future move it arrived at. Thus, tactics might be short, but they would have very large spreads.

Suppose the machine had some other goals in its memory labeled "opponent's goals". For example, the machine might have:

ctl (Q5).

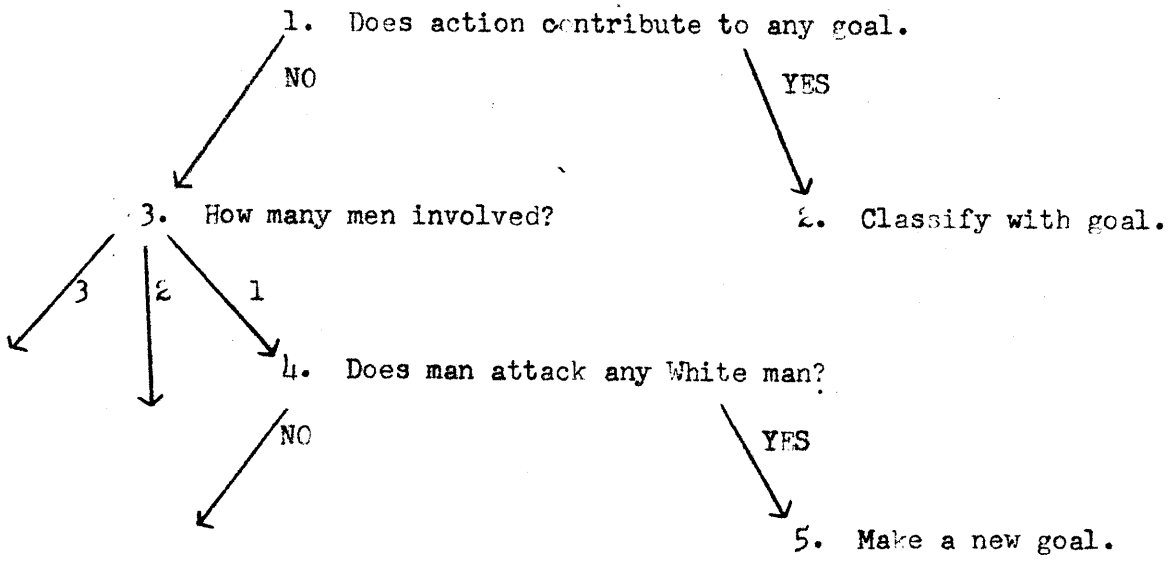
The machine interprets this as, "Control the Queen five square", and takes as a definition of achievement for the opponent that he could win an engagement if White tried to occupy the square or capture a Black man who did occupy it. If the machine considered this goal to be one of the opponent's important goals, then clearly, any opponent's action which achieved this goal would be relatively likely to occur. Or would it? It certainly depends to what degree the machine's representations of the opponent's goals correspond to the opponent's real goals or whatever other mechanisms guide his actions.

The machine must now be equipped with all the apparatus for collecting evidence, forming hypotheses, verifying them, and making inferences from them. At least in some rudimentary form these will all be required to develop and maintain a running model of the opponent's intentions.

If such a set of mechanisms were available to the machine, then, within

certain limits of accuracy, it could assign likelihoods to the various alternatives by inference from the model. These likelihoods provide a solution to half the problem of consequences: how to know which consequences to examine at any position that is the opponent's move. Examine those alternatives which seem most likely.

A rather large number of mechanisms are involved. First, each action that the opponent makes must be classified as stemming from some goal or goals. Somewhere in memory let there be stored some expressions like:



These expressions are coded in the language spoken of earlier. The machine starts with expression 1 and interprets it. This instructs the machine to test each of the opponent's goals to see if the action contributes to it. This term is used in a narrow sense: only men who can capture contribute to an attack; only men who command the square contribute to its control. Thus, in the example, "control Queen five," the machine would deter-

mine if the action had resulted in any new man commanding the Queen five square. If the answer is yes, the machine proceeds to expression 2. When interpreted, this instructs the machine to classify the action as stemming from the goal to which it contributes. This ends the decision process. But as usual, the answer could also be no; the first rule does not suffice to classify the action. The machine then proceeds to expression 3, interprets it, and counts the number of men who changed in the current action. Finding the number of men who changed in the current action. Finding the number to be one, which means that nothing complicated like a capture, promotion, or castle occurred, it proceeds to expression 4. The machine is now instructed to see if the man which moved can be considered as attacking some new man, and if so to read expression 5, which instructs the machine to create a new goal. Notice that this alternative is considered only if the action has failed to be consistent with any of the goals already hypothesized as accounting for the opponents action.

So far only a mechanism for classification and possible introduction of new goals has been described. It should be remarked about this mechanism that the content of the expressions is much less important at this juncture than the general schemat. Other expressions and more of them could just as well be used. Instead of describing the rest of the mechanisms needed to complete this component, it is necessary to move on to other key pieces. The undescribed elements will be constructed in the same vein. For instance, there will be expressions for obtaining likelihoods like, "How often has a goal been contributed to recently," and "High activity implies continued activity is likely."

Utility

Even admitting the machine can assign likelihoods to the opponent's alternatives, this solves only the half of the problem of consequences associated with opponent's moves. There remains the other half: how to know which of the machine's possibilities to explore when searching for a tactic.

Consistent with the design philosophy emerging here, the machine will have a section of memory devoted to expressions in the language that are relevant to this problem. These will be organized into a decision net in the same fashion as the expressions for classifying opponent's actions. The object of such a net will be to assign a utility to future machine actions that could be explored in trying to complete tactics. This utility would be expected to reflect a complex of factors. The feature to note is not just the possibility of such nets, but the kinds of relevant information that now are available to be used.

The most important kind, I think, is that every man has associated with it the functions that it is performing. These associations occur by means of the goal structure. Suppose there is a subgoal that the King-Knight supports the King-Bishop; that is, can retaliate by recapturing any man that captures the Bishop. This serves as a statement that the Knight has a function to perform. Any goal whose tactic depends on moving the Knight into a new position must compete with the goal which already has the Knight supporting the Bishop. The machine can determine this by a simple search of its goals. Therefore a reasonable decision net would recommend exploration of this Knight's moves only as a last resort.

The vision of a large collection of small reasons why various moves

should have low utility for exploration, raises a caution. We do not want the machine to spend all its time examining the future actions of committed men; yet if it were never to do this, it could overlook real opportunities and might even gradually paralyze itself by the accumulation of commitments. The solution, of course, is the random element. The difficulty with random search in a complex environment is that in reasonable time scales, nothing will ever be discovered. Therefore, randomness should be introduced into the decision nets only in small and well controlled amounts. The machine should rarely search for combinations which sacrifice a Queen and Rook to no apparent gain only to develop a mate eight moves later.

Level of Aspiration

By now we have finally returned to the decision problem stated earlier. There are devices for stopping the search for tactics if successful. There are devices for differentiating the various directions of search. These are likelihoods for the opponent's moves, and utilities for the machine's. But there exists no stop rule in the face of continued difficulties.

To find a solution we turn to a phenomena well known in the human sciences: levels of aspiration. The fundamental reason why the machine must terminate the search is limitation of resources, in this case mostly computing time. By a level-of-aspiration type solution, is meant the introduction into the machine of the limitation of resources as an explicit mechanism. That is, various expressions will exist in the machine which measure and utilize the information about the amount of resources available. Thus, the limitations will no longer function as absolute constraints; the way in which they affect the machine is partially within the machine's control.

For tactics a suitable measure, correlated with computing time but more

appropriate to the task, is the likelihood of a position. As the machine explores into the future, selecting one possible action after another, the likelihoods compound like probabilities so that the ultimate likelihood diminishes as actions get further into the future, or are reached through lower likelihood actions.

The machine sets out on a tactic search with a predetermined level of final likelihood. It gradually extends the tree of actions until each terminal position has fallen below this level; then it stops. If now, the aggregate likelihood of reaching a position of goal achievement lies above another predetermined aspiration level, the machine considers the tactic adequate. If not, it returns to the goal structure and develops some additional goals, as described earlier.

Transformation Rules

Nothing has yet been said about how all these sub-goals are generated. The structure of the mechanism is clear; it will consist of more rules of thumb. There will exist a set of expressions in the language which function like the rules of inference in logic: they allow us to derive new expressions from old ones. They are of the form, "For a goal of type A, try a subgoal of type B." For example, "For att (x,y) try def (y)." This expresses the fact that there is some merit to defending the men who are involved in carrying out an attack. If this transformation rule, as it is called, were applied prior to the opponent's actual attack on the attackers, this would constitute a fine example of anticipatory response.

The machine must examine its goal structure to determine the types of goals for which it requires new subgoals. It selects out those transformation rules which might be relevant; that is, where the structure of

the goals in the net fits the specifications of the first part, or premise, of the transformation rule. Several possible rules may be obtained. It seems appropriate to introduce a random element to guide the final choice, since the inference expressed by the rules is rather a loose one. The weights assigned to each rule will be functions of experience, so that rules which work get chosen relatively often.

We have again arrived at a place where the important question is the actual content of these expressions. Certainly, if the rules are poor the machine will play terrible chess. Its operation will bear little relation to the objective game situation. Again, other problems are more important. No solution has yet been described for either the evaluation or the alternatives problem. It will be appropriate to consider evaluations next.

Evaluations

It may seem that somehow the evaluation problem has an implicit solution in the array of mechanisms postulated so far. The problem is to relate a position to the winning of the game. Since the machine uses intermediate goals, the problem now is to relate positions to these sub-goals. It seems that the tactics do provide this relation. If a sequence of actions in a tactic results, sometime, in the given position, then the likelihood of achieving that tactic's goal from the position is known. The machine spent time computing it when generating the tactic.

The immediate difficulty with this is that it relates a position to a goal only if it occurs in that goal's tactics. This provides no evaluation for the other goals. Hence, it is only a partial solution since the problem of evaluation is assessing the relation to the ultimate goal, which is winning. In order to make playing possible, the machine has replaced

the single remote goal with a large number of more immediate ones. Thus, for the machine the problem is: for any given goal, determine the likelihood of achievement from any given position.*

The solution exists for special case where the position occurs in the tactic of the goal. Consider two other special cases. First, suppose the goal were achieved for the given position. It would receive a likelihood of one if we think of likelihoods as similar to probabilities. Second, suppose a goal were not achieved, had no tactic, and no subgoals. It would receive a likelihood of zero, since its achievement rests solely with coincidence or the opponent, both equally bad bets.

These special cases provide the basis for a solution the evaluation problem. If the machine generates goals as described, then at any instant all the terminal elements belong to one of the three special cases. This is so since the situations not covered by the special cases all involve a goal having subgoals, which means they cannot be terminal elements. The terminal evaluations are a set of boundary conditions, from which, step by step, evaluations can be assigned throughout the goal structure.

In the general case, the problem will be to compute the likelihood of a goal given the likelihoods of all its subgoals. It is here that the relation of subgoal is given operational significance. The machine operates

---

*
Throughout the paper I have very carefully used the term, "likelihood," instead of "probability". The logical status of the entities referred to is not at all clear since the machine uses the measured likelihoods to determine action which in turn determines whether the move or position will ever occur; that is, what the likelihood "really" is.

as if the following principle were true: if g_1 is a subgoal of g_2, then an increase in the likelihood of achievement of g_1 produces an increase in the likelihood of achievement of g_2. It interprets its experience in this light, so that contrary instances are treated as implying the subgoal is a poor one.

The laws of combination of likelihoods must have several simple properties, mostly those implied by the principle above. Other than this it may not make too much difference; simple linear combination may do admirably.

Although some of the details have not been enumerated, the machine now has the mechanisms for evaluation. The result is a large set of numbers, one for each goal. The reason why more combination is not required, or perhaps not even desirable, leads to the problem of alternatives.

Alternatives

Clearly, the machine is interested only in alternatives that further its goals. Now, goals are general statements, and the tactic is the bridge provided to pass between the goals and the very particular current situation. Tactics provide actions that lead to goal achievement. At any moment, then, there are a certain number of tactics and these yield an equal number of alternatives (not necessarily all distinct), all of which are worth considering since they each further some part of the machine's goal structure.

This would be a solution to the alternatives problem if this set were always adequate. It can hardly be expected to be so at all times. For instance, right after a very unexpected move by the opponent there may be no tactics left at all.

The solution lies in more levels of aspiration, partial efforts and

iteration. For each alternative, the machine computes the goal evaluations for the new position that would follow if the alternative were chosen. This yields an evaluation of each alternative's effect throughout the goal structure. Although this evaluation appears to be only one move deep, it is fundamentally grounded in the tactics, which extend much further into the future.

The decision is now made whether to make the available set suffice, or whether to return and work some more: to add and modify the goals and tactics. This is a level-of-aspiration type decision, which will depend not only on whether the alternatives are "good enough", but also on how much time remains, and whether the move is crucial. Only if the decision is made not to explore and expand further, is the best alternative picked from the limited set and punched into the card as the machine's actual move.

The term, "best alternative", is used in a very casual way. The evaluations consist of many numbers, at least one for each goal. It is clear that if a single alternative dominates all others, it should be chosen. It is also fairly clear that an alternative which achieves a very important subgoal is to be preferred over one which only increases the likelihood of a few very subordinate ones. But basically this is a multiple value situation, and in general no such simple rules can be expected to indicate a single best action. The problem for the machine is not to somehow obtain a magic formula to solve the unsolvable but to make a reasonable choice with least effort and proceed with more productive work. There are other ways to deal with the problem; for instance, include conflict as a fundamental consideration in the decision to explore further.

Thus at each move, the machine can be expected to iterate several times until it achieves an alternative that it likes, or until it runs out of time and thus loses the game by not being smart enough or lucky enough.

Performance Schemat

The pieces now exist to give an over-all schemat for the performance system of the chess learning machine. This is a set of mechanisms which is sufficient to enable the machine to play chess. There is no learning in this system; it will play no better next time because it played this time. If the content of all the expressions required is appropriate, it will play good chess; if they are not, it will play very poor chess.

This performance system is highly adaptive. A goal structure peculiar to each play of the game is generated during the course of play. Tactics reflect the minute detail of the current situation. This short run adaptability is not to be confused with learning which would permanently affect the way the machine would play in the future.

(Insert Fig. 1 about here.)

Figure 1 gives the schemat of operation. Rather than present as systematic and complete a representation as possible, attention has been given to relating the elements discussed so far. The rectangles represent the major kinds of information in the system. These may be viewed as memories. The arrows indicate processes that operate on one kind of information to produce another. The small writing by these arrows relates these processes to key words used earlier. Some of the main decisions are put in circles, since it makes the diagram easier to follow. The programs for carrying out most of these processes are the various nets, like the classification net. For the sake of clarity, these are not shown as explicit kinds of information,

although they certainly occupy a large part of the computer's memory.

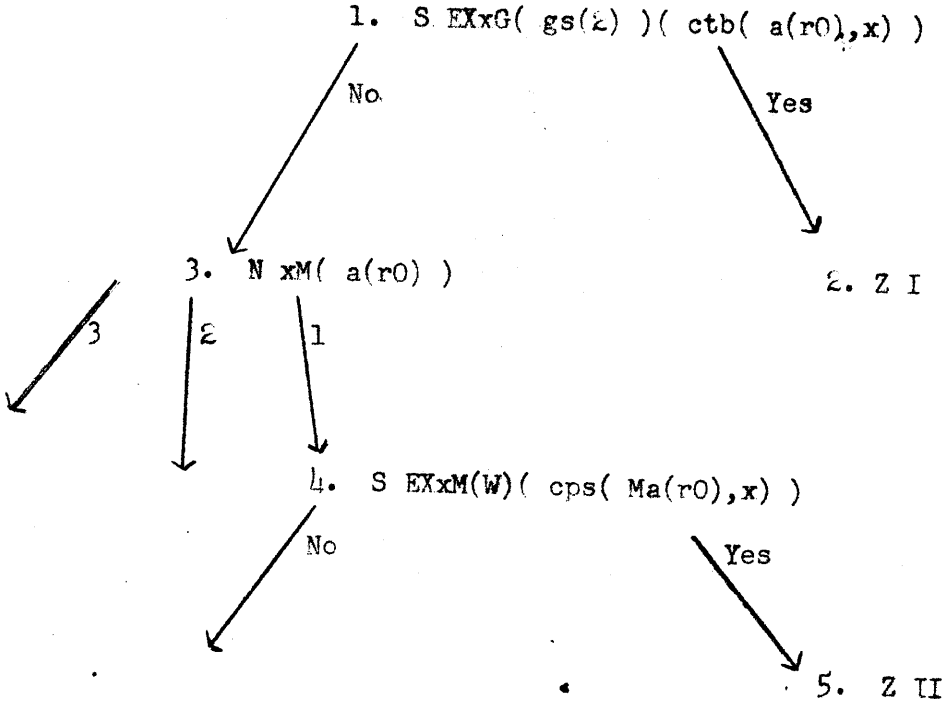
Each sequence always starts with an opponent's move being received (at the top). The process continues (downward) by a series of straightforward computations until the question is reached whether the situation is "good enough". This is the fundamental question. If the answer is yes, the machine has only to choose from among the available alternatives and play, thus ending the sequence (down). So far the effort spent is nominal. If, however, the answer is no, the machine proceeds to the modification and extension of the goals and tactics (to the right and up). This part is of indeterminant duration and effort and utilizes all of the complex apparatus that has been built up. Following this, the machine again attempts to produce a move (downward again). This is the fundamental cycle: to try to decide on a move with little effort, to modify the basis of decision, and to try again. Finally, of course, a move is made and the sequence stops.

Language

It is necessary to add a few comments about the general purpose language required to make a machine of this nature feasible. An essential feature of this language is its ability to refer to itself as well as to the "external" chess situation. The only languages of this power of expression that have been formalized at all are those used in symbolic logic such as the calculus of propositions (6). The one illustrated below is an adaptation for computer use.

Each symbol has a binary code, although for efficiency's sake, not all have different codes. An expression, then, is a long sequence of zeros and ones. The expression is decodable by proceeding from left to right: knowing which symbols have been found so far gives sufficient information to determine how many bits to consider next and what class of symbols it belongs to.

The following illustrates this language. It is the translation of the classification net exhibited earlier.



Consider expression 1 as an example. In toto it means, "Does the action contribute to any goal?" A question in the symbolic language is a sentence without its truth value. The machine interprets the missing value as an instruction to determine it; that is, to obtain the answer to the question. Expression 1, then, starts with 'S' to identify itself as a sentence. (Expression 3 has an 'N', and its value would be a number). "EX x G" means, "there exists an x which is a goal". The "gs (2)" in parentheses gives additional specifications on the range of x. The machine will only search gs(2) which is the opponent's goal structure. The last parentheses contains the predicate, "contribute" symbolized by "ctb".

Its arguments are first, an action, "a(r0)", the "r0" standing for relative time zero so the machine will always look at the current action; and then a goal, "x", which is the variable over which the machine will search.

The machine interpretation and execution of such an expression runs along lines very similar to those used in algebraic coding schemes. The machine has programs for determining the truth or falsity of each individual predicate, here only the "ctb". Truth values are combined, with due regard for brackets, by the laws of "and", "or", and "not", depending on which occur in the expression. It interprets "EX" as making the sentence true if for any of the goals considered it obtains 'true' for ctb.

Learning

The section is limited to some remarks on the requirements for learning and the potentialities for it that exist in the machine.

The large number of highly interrelated mechanisms involved in the performance of the machine indicates a major difficulty. Consider the hundreds of expressions which each determine a highly specific rule of action. How is it possible to have a set that actually fits together to produce effective chess play? The assumption that man can discern which sets will work seems rather untenable; there are too many effects and hidden consequences. Learning seems to offer a solution.

At any given time the machine has a set of expressions, which work tolerably well over some restricted range of environments. The machine generates a few extensions or modifications to its current set. These are incorporated if their use provides an improvement in performance. The features to note about this learning process are first, that it gradually extends the set of expressions and second, that expressions get admitted

only if they "fit in" with the others already there. Thus, given that an appropriate training sequence of games are played, the machine will grow itself a set of expressions of sufficient complexity to play good chess.

Many of the mechanisms necessary to provide this learning are already in the machine. First, it is necessary to measure the effects of one mechanism on another. The machine is already able to perform any test or measurement given by a network of expressions. All that is required is that the expressions refer to the appropriate internal parts of the machine and ask the right questions. The language is powerful enough to do this. These nets also allow diagnosis of which parts of a complex mechanism are causing the undesirable effects. Memory is needed to keep performance records, and mechanisms are needed to collect and classify these records; all of which are already possible. The possibilities for new mechanisms are limited only by the modes of expression of the language, since the specific behavior of the performance system is determined by the content of expressions. Finally, the search for new mechanisms is quite similar to obtaining subgoals for goals. That is, there would be sets of transformation rules to give the types of modifications that might prove useful for a given type of expression.

The problem of sample size requires mention: How large a sample of experience is necessary to obtain learning? Or better: How much information about the effects of behavior is necessary to successfully modify the behavior? Chess affords a good example of this problem. It is extremely doubtful whether there is enough information in "win, lose, or draw" when referred to the whole play of the game to permit any learning at all over available time scales. There is too much behavior. For learning to take place each play of the game must yield much more information. This is

exactly what is achieved by breaking the problem into components. The unit of success is the goal. If a goal is achieved its subgoals are reinforced; if not, they are inhibited. (Actually, what is reinforced is the transformation rule that provided the subgoal.) This is so whether the game is ultimately won or lost. Each play gives learning information about each goal that is generated. This also is true of the other kinds of structure: every tactic that is created provides information about the success or failure of tactic search rules; every opponent's action provides information about success or failure of likelihood inferences; and so on. The amount of information relevent to learning increases directly with the number of mechanisms in the chess playing machine.

Conclusion

"As every design engineer knows, the only difference between a good design and the actual machine is time and effort." This adage is a byword in the field of robots and thinking machines. The scheme prsented here is not far from a "good design." One can estimate the man hours necessary to draw up the detailed flow diagrams from which machine coding follows as routine chore. But this is not sufficient. These mechanisms are so complicated that it is impossible to predict whether they will work. The justification for the present article is the intent to see if <u>in fact</u> an organized collection of rules of thumb can pull itself up by its bootstraps and learn to play good chess.

References

1. Shannon, C. E. Programming a computer for playing chess. Phil. Mag. 1950 41 314 256-275

2. Weinberg, M. Mechanism in neurosis. Amer. Scientist. 1950 39 74

3. Richards, P. I. On game learning machines. Scientific Mon. 1952 74 4 201-205

4. Strachey, C. S. Logical or non mathematical programmes. Proc. Ass. Computing Machinery. Toronto. 1952 46-49

5. von Neumann, J. and Morgenstern, O. Theory of games and economic behavior (2nd ed). Princeton: Princeton University Press 1947.

6. Carnap, R. Logical syntax of language. New York: Harcourt, Brace, and Co. 1937

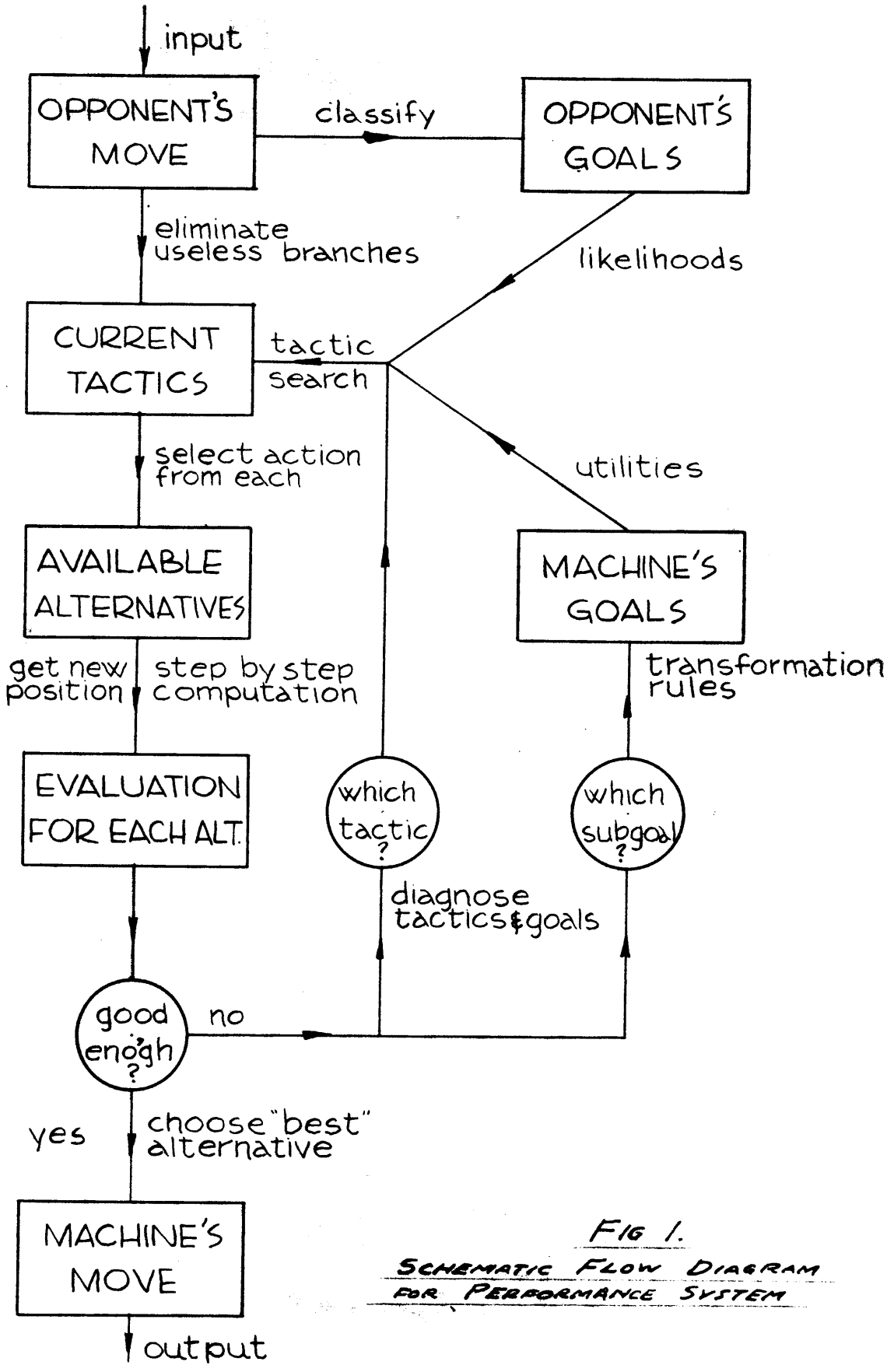


FIG 1.

SCHEMATIC FLOW DIAGRAM FOR PERFORMANCE SYSTEM