A STUDY OF TECHNIQUES FOR GAME-PLAYING PROGRAMS

T.A. MARSLAND and P.G. RUSHTON

Computing Science Department, University of Alberta, Edmonton, Canada

SUMMARY

An attempt has been made to describe the techniques used to play 'sophisticated' games, of which chess is taken to be representative. At present, many chess-playing programs use a polynomial evaluation function, in conjunction with forward pruning and the minimax backing up procedure, to play the game. These techniques are discussed along with their associated dif-difficulties. Some possible means for modifying the coefficients of the scoring function (which is equivalent to a form of learning) are outlined, but results indicate that they will not be successful. Finally, the specifications for a goal-directed approach based on feature recognition, goal suggestion and planning are proposed. This approach has greater potential for producing a program whose play is of the highest standard.

INTRODUCTION

In this paper the techniques used in various game-playing programs will be surveyed, and an assessment made of their utility and future extensibility. It will be explained why an evaluation function should not form the basis for a learning algorithm, and why the lack of continuity in the planning capability of contemporary methods makes them incapable of producing a program whose play is of the highest calibre (even if all reasonable time constraints are removed).

The essential features of games which cannot be solved by exhaustive means will be identified, and from these features the specifications of a sophisticated game-playing program will be developed. A goal directed procedure to meet those requirements will be discussed, and a partial assessment of the potential capabilities of this approach presented.

The sophisticated games which we consider (in particular Chess, Checkers and Go) are all mathematically trivial, because an exhaustive method exists for their play; this method is, however, totally impractical from a programming point of view. For example, in the game of chess it is estimated that there are 35^{40} positions (not all of which are unique) in the decision tree for the first 20 full moves from the start, so it would take an eternity to enumerate them all. Even for checkers the method is impractical. Other approaches must obviously be found, and some of these will be considered.

Most games may be split into three phases: opening, middle game and ending. Memory plays a very important part in two of them, because there are relatively few principal variations in the opening, and few basic strategies in the ending. In computer terms specialized programs are probably most appropriate in these two places. However, the middle game provides the greatest degree of complexity and offers the most opportunity for original behaviour by a program. We shall therefore confine our remarks to that phase.

Two ways of tackling the problem have been considered. One is evolutionary in that successively bigger, better and more specialized programs are written until acceptable solutions are found. The other relies on the implementation of general learning mechanisms to produce programs which can train themselves. As might be expected, the evolutionary approach has met with many early successes, but can only progress very slowly to the best (master) human level. General learning algorithms have not yet been developed, though some success has been achieved on simpler games [1].

CURRENT TECHNIQUES

Although there had been attempts to build mechanical game players, it was Shannon's article [2] which initiated the interest in chess-playing programs. One of his main ideas was to use an evaluation function, in conjunction with a min-max procedure, to build a reasonable game tree. Subsequent papers by Newell, Shaw and Simon describing work on a General Problem Solver and a chess program [3], Samuel's work with Checkers [4] and Good's article [5], are additional sources of fundamental ideas. The common theme behind these papers is the construction of a game tree (a structure of decision points, referred to as nodes). Practical restrictions ensure that only a partial tree can be built. The size of the tree is controlled in two ways: reverse pruning (the alpha-beta or M&N methods) and forward pruning. The reliability of the alpha-beta method

method is directly related to the precision with which the evaluation function measures the relative worth of the 'terminal' positions. These terminal positions occur either at some predetermined depth or at a quiescent node (at which the error in the scoring function is thought to be small). Forward pruning also occurs whenever a subset of the available continuations is explored, or if some unattractive variation is abandoned. In order to increase the number of alpha-beta cutoffs, the available moves at each node are considered in order of decreasing importance (value). If a relatively sophisticated evaluation function is employed, it may be possible to explore a subset of the available moves at each node. Dramatic reductions in tree size are obtained by this method, at substantial risk. Three main approaches to the tree-building process have been tried: -

1) The earliest programs employed an exhaustive search to a fixed depth, of say five ply (two and a half full moves) and this approach is again being successfully used by Gillogly in his TECHnology program [6]. Its intended use is to provide a lower bound on the expected performance of programs which use more advanced methods. Even using the standard pruning techniques, the complete tree to fixed depth produces an enormous number of terminal positions. Thus only a simple calculation, such as material balance, can be done to assess the relative merits of the continuations.

2) Some of the best examples of the approach suggested by Shannon are to be found in the series of programs, culminating in MAC HACK [7], which were written at MIT. These programs, and in fact most of those written in the 1960's, were based on the premise that a far smaller and more directed game tree can be constructed, provided more analysis is done at each decision point. The analysis takes the form of assigning values for features found in the specific game being played, from which an ordered list of preferred moves is generated.

Perhaps the major weakness with the evaluation function approach is in the preferred move generator, because by the very nature of the game-features it is easier to generate tactical moves, which are considered at the expense of positional ones. Positional moves can only be conceived as part of some long range plan. Botvinnik [8] suggests the use of a horizon (minimal length attack path) to help in the conception of these positional moves, but even his proposal seems to be of greater merit in forming tactics.

3) One other fundamentally different approach to the tree

construction problem is the contribution of the goal seeking methods, of Newell et al. Their chess-playing program studies were inconclusive, being de-emphasized in favour of GPS. Although the program handled the opening fairly well, it was not shown that its implementation could be successfully extended to the middle game, where the relevancy of conflicting goals has to be resolved.

The most important aspect of this scheme is that, of all feasible moves, only those which satisfy some predetermined goal are generated. If it is implemented properly, preferred moves and their game subtree will be analysed less expensively than with the other methods, because of the high degree of selectivity at each node. Other advantages are that more opportunity for continuity of a plan within a single variation is provided, and irrelevant moves are neither considered nor generated.

A major failure of many contemporary programs is that they treat each node in a continuation as an independent position. A move is generated at ply N without regard to the reasons which suggested the preceding move at ply N-2. There is, therefore, no deliberate continuity of plan within a given variation, which is much more serious than failing to preserve the subtree of the principal variation from move to move, since the salvaged subtree is very small in proportion to the size of the game tree that is constructed. Some continuity can be achieved by noting the problems which arise in a given variation. Increased importance is then given to those moves which counter the newly uncovered threats, by dynamically re-ordering the preferred move list at every decision point.

EXTENSIBILITY OF THE METHODS

Gillogly estimates that a 7-fold increase in computing power is needed to add an even ply layer to his tree, and somewhat less for an odd ply layer. Theoretical bounds for the cost of adding layers are given by Slagle and Dixon [9]. The most likely source of this speed increase is from advances in parallel processors, but the exponential growth in the number of terminal nodes eliminates any hope of using a complex evaluation function.

Polynomial scoring functions with adjustable weights (coefficients) are fundamentally unsound, being unreliable. They can give neither an absolute measure of a position's worth nor a precise relative assessment. When used to select the order in which continuations are explored, the scoring method is akin to employing a maximum gradient scheme with fixed step size to search a nonconvex surface.

One supposed advantage of the polynomial evaluation function is that its weights may be adjusted (or 'learned') to improve its selectivity. Given a set of positions and their corresponding best (master) moves, Slagle [10] suggests several ways of modifying the weights, to increase the number of correct selections made by the scoring function. Under the assumption that there is no perfect scoring function, a directed tree search is necessary. Thus it may be more satisfactory simply to find a set of weights which places the master's move within some window (of say 5 moves) at the top of the preferred move list. Some more complicated schemes have been tried, but no significant improvement can be reported. For example, Fig. 1 compares the selectivity error rate of a six variable polynomial, whose weights were generated by the correlation coefficient method, an optimization method and an experienced player, over 378 positions selected from 15 master games. Perhaps the most important observation is that, for a 5-move window only an 80% selectivity was obtained with this polynomial. In order to develop a master calibre program, the preferred move generator must have 100% selectivity. It is unlikely that this requirement can be met by an evaluation function, with an acceptably small window.

One might hope that by adding terms to the polynomial a progressively more advanced player would be developed. In practice such is not always the case, since there is increased possibility that an unexpected move will be generated for the wrong reason, when an accumulation of errors cancel some important feature.

The whole concept of a weighted sum of features, although computationally satisfying, would appear to be different from the way in which de Groot has found that humans play [11]. One now has to decide whether game-playing problems can be reformulated in a manner which is amenable to computer solution, or whether computer hardware/software must be designed to simplify the imitation of human strategies. De Groot has observed how chess players perform over the board analysis. He noted that they appear to work in the following stages: -

1) Analysis of very few basic continuations.

2) Re-examination of variations in increasing detail.

3) Selection of a move on a basis of one or two overriding features. Even though many factors are considered, no weighted sum of components is computed nor are minor strengths or weaknesses in a move allowed to cancel more important features.

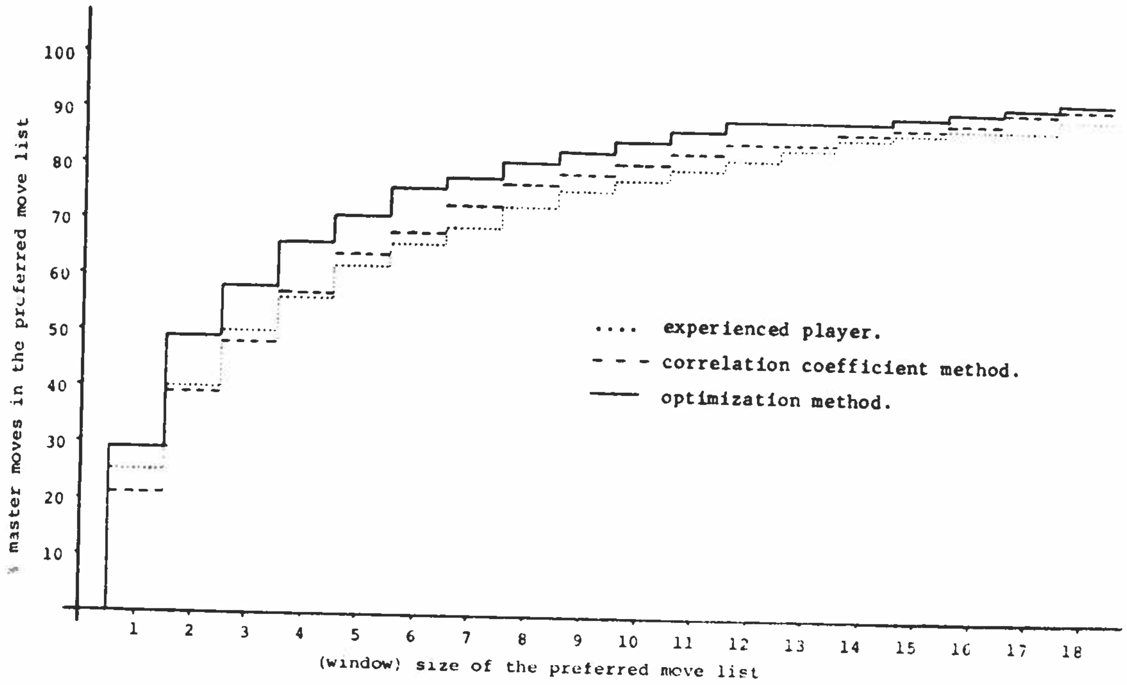

Fig. 1. Cumulative distribution of master moves in the preferred move list.

It would seem that the fact that relatively few features dominate the selected move can be more easily exploited by the goal-seeking methods, as will be shown later. Even so, it does not necessarily follow that programs to play these games must model human behaviour, but rather that our ability to employ the computer's facility for rapid execution of repetitive algorithms has so far proved inadequate.

GOAL SEEKING APPROACH

The general hope with the evaluation function method is that if nearly all the locally important moves from each decision point are explored, there is high probability that the globally best (within the limits of the tree) continuation will be found. Since the method has no known destination, it has to assess the relative merits of the terminal positions. There is not much advantage in making this assessment with a function which contains more features than the one which generated the preferred move list, since the tree growth has not been controlled (biased) to develop nodes that exhibit those extra properties.

With the goal seeking methods it is anticipated that by specifying a desirable goal, and only generating those moves which are relevant to that goal plan, one can narrow the search task. The opponent's moves are restricted to those which either inhibit one's plan or help him achieve a more desirable goal. The method therefore hinges on accurate recognition of attainable goals, that is, accurate predictions of desirable global continuations. If successful, one is assured that all terminal positions are valuable.

The approach we are exploring has three distinct phases: -

1) Feature recognition and evaluation.

2) Goal list construction.

3) Planning.

All of these phases draw upon the knowledge state, which consists of the current position, the current plans and the status of the relevant features that have been found so far.

The feature recognition programs receive the current knowledge state as input and provide a measure of the degree of presence of specific features. For chess typical features might be:
<weak King> ==> <badly castled> OR <K held in centre> OR ...
<badly castled> ==> <castled K-side> AND <weak K-side pawns> OR ..
<K held in centre> ==> <K in centre> AND <K unable to castle>.

All the features are game specific, more advanced ones being built up from such primitives as material balance, control of key squares, and under/over committed pieces. Goal statements are constructed in terms of these features, as illustrated by: ACHIEVE <centre control> BUT NOT <double pawns> UNLESS <material advantage>. The goal statements are now passed on to the planner, whose job it is to break each component of the goal statement into more fundamental subgoals. This process is continued until a subgoal becomes a piece configuration for which a move can be generated. The terms in the goal statement are ordered, so that a simple tree structure can be built to help explore the subgoals. During the planning stage, weaknesses in the initial goal statement may be discovered; the planner modifies the statement to correct those deficiencies.

Although there will initially be a fixed list of feature recognition procedures, not all of them will be active at one time. There will be feature management routines, working during the opponent's thinking time, to eliminate searching for irreversible features like castling. Feature procedures could be removed from the active list on a least recently useful basis, allowing the activation of such endgame features as pawn promotion. In summary: -

1) The knowledge state consists of the board description and the current goal plan.

2) This plan allows information to be gathered about the position, by the execution of the active feature procedure.

3) Initial goal statements are formed which seek to preserve the extant good features and eliminate the bad, and existing features of the opponent are negated. Desirable goals that could be achieved from this position are also added.

4) The planning programs now elaborate the goal statements, until a time limit expires.

5) A preferred move list is generated. At this stage such factors as the length of the move sequence and the number of goals that the moves satisfy will be taken into account. If no moves are proposed a standard evaluation function technique will be applied.

6) Once a move is made, the goals and the elaborations which caused its generation will be added to the knowledge state, allowing continuity of plan from move to move.

CONCLUSIONS

From our studies of games we have concluded that a master calibre program cannot be written on a basis of either building a complete tree to a fixed depth or using an evaluation function to build a tree from a subset of the available moves. The first method cannot afford an adequate evaluation function, while the second is in grave danger of building an irrelevant tree, since it cannot be certain that the master move is within its window. The goal directed method, although technically more difficult to implement, is well established as the basis for successful algorithms, since it is the product of human evolution.

Two deficiencies of most contemporary programs are that they cannot generate either apparently sacrificial or "quiet" positional moves. A return to the goal-seeking approach is suggested, in the manner outlined. Not only is this approach capable of overcoming the weaknesses of the evaluation function method, but it also allows greater potential for the implementation of learning algorithms, especially in the area of feature creation.

REFERENCES

1. N.V. FINDLER, "Some new approaches to machine learning", *IEEE Transactions on Systems Science and Cybernetics*, 173-182 (July, 1969).

2. C.E. SHANNON, "Programming a digital computer for playing chess", *Philosophical Magazine 41*, 356-375 (March, 1950).

3. A. NEWELL and H.A. SIMON, *Human Problem Solving* (Prentice-Hall, New York, 1972) pp. 678-698.

4. A.L. SAMUEL, "Some studies in machine learning using the game of checkers, II: recent progress", *IBM J. Res. Dev.* 601-617 (Nov. 1967).

5. I.J. GOOD, "A five year plan for automatic chess" *Machine Intelligence 2* (American Elsevier 1968) pp. 89-105.

6. J.J. GILLOGLY, "The technology chess program" *Report CMU-CS-71-109* (Dept. of Computer Science, Carnegie-Mellon Univ. Nov. 1971).

7. R.D. GREENBLATT, D.F. EASTLAKE III and S.D. CROCKER, "The Greenblatt chess program" *AFIPS Conf. Proc. 31*, 801-810 (1967).

8. M.M. BOTVINNIK, *Computers, Chess and Long Range Planning* (Springer-Verlag, Berlin, 1970).

9. J.R. SLAGLE and J. DIXON, "Experiments with some programs that search game trees" *J. ACM* (April 1969).

10. J.R. SLAGLE, *Artificial Intelligence: The Heuristic Programming Approach* (McGraw-Hill, New York, 1971) pp. 143-162.

11. A.D. DE GROOT, *Thought and Choice in Chess* (Mouton 1965).