Using Patterns and Plans in Chess

David Wilkins
SRI International, Menlo Park, CA, U.S.A.

ABSTRACT

The purpose of this research is to investigate the extent to which knowledge can replace and support search in selecting a chess move and to delineate the issues involved. This has been carried out by constructing a program, PARADISE (PAttern Recognition Applied to DIrecting SEarch), which finds the best move in tactically sharp middle game positions from the games of chess masters. It encodes a large body of knowledge in the form of production rules. The actions of the rules post concepts in the data base while the conditions match patterns in the chess position and data base. The program uses the knowledge base to discover plans during static analysis and to guide a small tree search which confirms that a particular plan is best. The search is "small" in the sense that the size of the search tree is of the same order of magnitude as a human master's search tree (tens and hundreds of nodes, not thousands to hundreds of thousands as in many computer chess programs).

Once a plan is formulated, it guides the tree search for several ply and expensive static analyses (needed to analyze a new position) are done infrequently. PARADISE avoids placing a depth limit on the search (or any other artificial effort limit). By using a global view of the search tree, information gathered during the search, and the analysis provided by the knowledge base, the program produces enough terminations to force convergence of the search. PARADISE has found combinations as deep as 19 ply.

1. Introduction

One of the central concerns of artificial intelligence is expressing knowledge and reasoning with it. Chess has been one of the most popular domains for AI research, yet brute force searching programs with little chess knowledge play better chess than programs with more chess knowledge. There is still much to be learned about expressing and using the kind of knowledge involved in playing chess. CHESS 4.7 is probably the best chess program, and its authors make the following comments in [12]:

> But why is the program so devoid of chess knowledge? It is not because we are deliberately trading it for speed of execution.... It is not quite true to say that we don't know how to feed additional knowledge to the program.... Our problem is that the programming tools we are presently using are not adequate to the task. Too much work is needed to encode the sizable body of knowledge needed to significantly improve the quality of play.

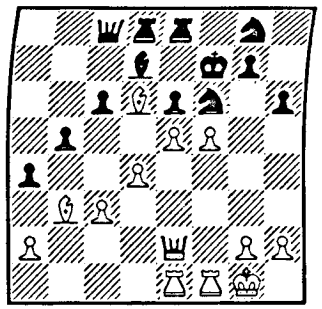
Human masters, whose play is still much better than the best programs, appear to use a knowledge intensive approach to chess (see [2]). They seem to have a huge number of stored 'patterns,' and analyzing a position involves matching these patterns to suggest plans for attack or defense. This analysis is verified and possibly corrected by a small search of the game tree. Unlike the searches conducted by most programs, the human master's search usually has specific goals to accomplish and returns useful information which affects the analysis of other lines. The purpose of this research is to investigate the issues involved in expressing and using pattern-oriented knowledge to analyze a position, to provide direction for the search, and to communicate useful results from the search.

To reduce the amount of knowledge that must be encapsulated, the domain of the program has been limited to tactically sharp middle game positions. The phrase 'tactically sharp' is meant to imply that success can be judged by the gain of a material rather than a positional advantage. The complexity of the middle game requires extensive search, providing a good testing ground for attempts to replace search with knowledge and to use knowledge to guide the search and communicate discoveries from one part of the tree to another.

Since, in our chosen domain, the attacking side can generally win material with correct play, the program uses different models for the offensive and defensive players during its search of the game tree. The offensive player uses a large amount of knowledge, suggests many esoteric attacks, and invests much effort in making sure a plan is likely to work before suggesting it. The defensive player uses much less knowledge and effort, tries any defense that might possibly work, and tries only obvious counter attacks. Trying every reasonable defense enables the offense to satisfy itself that a line really does win. The size of the tree is kept small because the offensive player finds the best move immediately in most cases.

For a knowledge based program to achieve master level performance, it seems necessary that the knowledge base be amenable to modifications and additions. PARADISE provides for easy modification of and addition to its knowledge base (without adversely affecting program performance), thus allowing the knowledge to be built up incrementally and tuned to eliminate errors. This is accomplished by having the knowledge base composed of largely independent production rules which are written in a straightforward Production-Language [13].

The goal is to build an expert knowledge base and to reason with it to discover plans and verify them with a small tree search. As long as this goal is reached, the amount of execution time used is not important. It is a design decision to sacrifice efficient execution in order to use knowledge whenever reasonable. When a desired level of performance is reached, the program could be speeded up considerably by switching to a more efficient representation at the cost of transparency and ease of modification.



1. Q–R5ch N×Q
2. P×Pch 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. R×Nch K–R4
9. P–N3 any
10. R–R4 mate

FIG. 0. White to move.

Because it grows small trees, PARADISE can find deeper combinations than most chess programs. The position in Fig. 0 is an example of such a combination from an actual master game. White can mate by sacrificing his queen and playing nine additional moves (against black's line of longest resistance), the next to last move not being a check or capture. Thus the search tree must go at least 19 ply deep to find this combination. This is considerably deeper than any other current chess program could probe in a reasonable amount of time, but PARADISE solves this problem after generating a search tree of 109 nodes, in 20 minutes of CPU time on a DEC KL-10. (PARADISE is written in MacLisp.) The fact that PARADISE can accurately analyze all relevant lines in only 109 nodes shows that the program makes good use of its knowledge.

2. Overview of PARADISE

PARADISE uses a knowledge base consisting of about 200 production rules to find the best move in chess positions. Every production has a pattern (i.e., a complex, interrelated set of features) as its condition. Each production can be viewed as searching for all instances of its condition pattern in the position. For each instance found, the production may, as its action, post zero or more concepts in the data base for use by the system in its reasoning processes. A concept consists of a concept name, instantiations for a number of variables, and a list of reasons why the concept has been posted. The data base can be considered as a global blackboard where productions write information. The productions are built up by levels, with more primitive patterns (e.g., a pattern which matches legal moves) being used as building blocks for more complex patterns. A search tree (the major component of most chess programs) is used to show that one move suggested by the pattern-based analysis is in fact the best.

Fig. 1 shows one of the simplest productions in PARADISE. It attempts to find an offensive plan which attacks a trapped defensive piece. The first three lines in Fig. 1 constitute the pattern or condition of the production rule while the

```
((DMP1)
(NEVER (EXISTS (SQ) (PATTERN MOBIL DMP1 SQ)))
(NEVER (EXISTS (P1) (PATTERN ENPRIS P1 DMP1)))
(ACTION ATTACK ((OTHER-COLOR DMP1) (LOCATION DMP1)) (THREAT (WIN DMP1))
(LIKELY 0)))
```

FIG. 1. Production rule in PARADISE which recognizes trapped pieces.

fourth line constitutes the action. The first line specifies one variable that must be instantiated. Because of its name, the variable must be instantiated to a defensive piece that is not a pawn. The next two lines in the production access two more primitive patterns that have already been matched, MOBIL and ENPRIS. The MOBIL pattern matches each legal move that can be made without losing the moving piece. If there are no squares which match MOBIL when DMP1 is the first argument, then DMP1 is trapped. Only in this case will there be any possible instantiations for DMP1 after the second line in Fig. 1 is matched. The third line prevents the pattern from matching if the trapped piece is a ready en prise. The idea is that in this case DMP1 should be captured outright rather than attacked. Thus this production matches only trapped pieces which are not en prise.

In Fig. 1, the action of the production rule posts an ATTACK concept. This concept gives instantiations for two variables and tells the system that the opposite color of DMP1 would do well to attack the square which is DMP1's location. In addition one reason is given for suggesting this concept. This reason consists of attributes describing the threat of the concept and how likely it is to succeed. Concepts are discussed in more detail later in this paper. The more primitive patterns (such as MOBIL) and the language in which productions are written are described in detail in [13].

To offset the expense of searching for patterns in the position, problems must be solved with small search trees. In fact, PARADISE can be viewed as shifting some of its search from the usual chess game tree to the problem of matching patterns. PARADISE shows that its pattern-oriented knowledge base can be used in a variety of ways to significantly reduce the part of the game tree which needs to be searched. This reduction is large enough that a high level of expertise can be obtained using a large but still reasonable amount of resources. This reduction is also large enough that PARADISE avoids placing a depth limit (or any other artificial effort limit) on its search. The various ways in which the knowledge base is used are briefly mentioned below.

Calculating primitives. PARADISE's knowledge base contains about twenty patterns (productions without actions) which are primitives used to describe a chess position. These patterns are matched once for each position. They range from matching legal moves to matching patterns like MOBIL and ENPRIS. The primitives are relatively complex and calculating them is expensive.

Static analysis. Given a new position, PARADISE does an expensive in-depth static analysis which uses, on the average, 12 seconds of cpu time on a DEC KL-10. This analysis uses a large number of productions in the knowledge base to look for threats which may win material. It produces plans which should guide the search for several ply and should give information which can be used to stop the search at reasonable points.

Producing plans. The knowledge base is used to produce plans during static analysis and at other times. Through plans, PARADISE uses its knowledge to understand a new position created during the search on the basis of positions previously analyzed along that line of play, thus avoiding a static analysis. Plans guide the search and a static analysis is only occasionally necessary in positions created during the search. Producing a plan is not as simple as suggesting a move, it involves generating a large amount of useful information to help avoid static analyses along many different lines and to help detect success or failure of the plan as it is executed. (Examples are given later in this paper.)

Executing plans. The execution of a plan during the tree search requires using the knowledge base to solve goals specified in the plan. This use of the knowledge base could be viewed as a static analysis where the system's attention is focused on a particular goal or aspect of the position. Such a focused analysis can be done at a fraction of the cost of a full analysis which examines all aspects of the position.

Generating defensive moves. PARADISE does static analyses and executes plans only for the offense (the side for which it is trying to win material). To prove that an offensive plan succeeds, the search must show a winning line for all reasonable (i.e., any move which might be effective) defensive moves. Productions in the knowledge base are used to generate all the reasonable defenses.

Quiescence searches. When a node is a candidate for termination of the search, PARADISE uses a quiescence search to determine its value. This quiescence search may return information which will cause the termination decision to be revoked. Productions in the knowledge base provide much of the knowledge used in PARADISE's quiescence search. Quiescence searches in PARADISE investigate not only captures but forks, pins, multi-move mating sequences and other threats. Only one move is investigated at each node except when a defensive move fails.

Analysis of problem upon failure. When either the defense or offense has tried a move that failed, the search analyzes the information backed up with the refutation in an attempt to solve the problem. The knowledge base is used in this analysis to suggest plans which thwart the opponent's refutation. In this way PARADISE constructs new plans using the information gained from searching an unsuccessful plan.

Answering questions. The search often requests certain kinds of information about the current position. The knowledge base is used to provide such information. The most frequent requests ask if the position is quiescent and if there are any obviously winning moves in the position.

The above list describes eight general functions the knowledge base performs for PARADISE. This paper explains how plans are produced and executed. The productions, the static analysis, the tree search, and mechanisms for communicating information from one part of the tree to another are discussed in [13].

3. The Need for Concepts

To understand a chess position, a successful line of play must be found. Human masters understand concepts which cover many ply in order to find the best move. A knowledge based program must also have the ability to reason with concepts that are higher level than the legal moves in a chess position. The following position from [1] illustrates this.

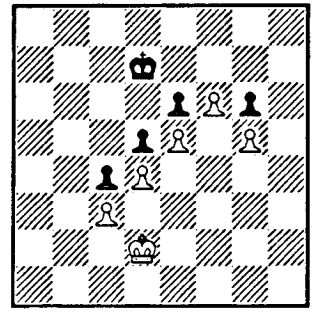


FIG. 2. White to move.

In the above position, most computer chess programs would search the game tree as deep as their effort limit allowed and decide to play K–K3 which centralizes the king. They cannot search deep enough to discover that white can win since white's advantage would take more than 20 ply to show up in most evaluation functions used by computer programs. The point is that the solver of the above problem uses conceptual reasoning to solve it, and the concepts must be higher level than the legal moves in a chess position. A human master would recognize the following features (among others) in the above position: the black king must stay near his KB1 to prevent white's king bishop pawn from queening, the pawns form a blockade which the kings can't penetrate, and the kings can bypass this blockade only on the queen rook file.

For a knowledge based program to solve this problem, it must be able to express concepts at the level of conceptualization used above. The system must combine and reason with these concepts in unforeseen ways. In the above example, the blockading concept would be used to develop the plan of moving the white king to the queen rook file in order to bypass the blockade, but if both black and white had pawns on their QN4 then the blockading concept would be used to decide that neither king can invade the other's territory. Expressing and using concepts is a major problem for a knowledge-based chess program.

To clarify the problems involved in used concepts to reason about chess, a comparison to a well-known 'expert' system such as MYCIN is helpful. The following phrase is the action part of a typical MYCIN production rule (from [4]):

> *then there is suggestive evidence (0.7) that the identity of the organism is bacteroides*

Such an action effectively adds to the plausibility score for some possible answer. This presupposes that any one writer of all the rules knows every possible solution since such rules will never create a diagnosis that has not been mentioned in the action part of some rule.

Using actions similar to those used in MYCIN rules would be similar to a chess system where the action part of each rule was to add to the plausibility score of either

(1) one or more of the legal moves in the position, or

(2) some prespecified object which could lead (through other rules) to recommending a legal move.

Such an approach is not satisfactory for the type of chess reasoning needed in the above problems. Concepts at a higher level than legal moves must be used in this reasoning, and objects cannot be prespecified (e.g., the extent of blockade must be created dynamically). Thus, PARADISE must reason by linking concepts to create plans instead of filling in prespecified slots.

In most current expert production systems (such as MYCIN), the action parts of the rules are in some sense part of the solution or a pointer to something which can produce a solution. In chess the system must discover lines of play that were not implicit in the knowledge base. Thus the word 'create' is used. MYCIN has a small solution set (120 organisms) and can mention each solution in its rules, while a chess program must discover an enormous number of lines using a set of productions which is much smaller in size (by an order of magnitude or more) than the number of plans it must recognize. This can be seen by looking at the actual production rules: MYCIN's actions mention the names of objects while PARADISE's actions involve many variables whose instantiation is not determined until execution of the action.

4. Knowledge Sources and Concepts

When a production in PARADISE matches, it may have an action part which posts various concepts in the data base. The program is able to express and use concepts by dividing its knowledge base into various *Knowledge Sources*. Each Knowledge Source (KS) provides the knowledge necessary to understand and reason about a certain abstract concept. In its simplest form, a KS is a group of productions which knows about some abstract concept, and a list of variables such that an instantiation of the variables represents a specific concept which corresponds to this abstract concept. For example, the production in Fig. 1 posts an ATTACK concept which corresponds to the ATTACK KS. The ATTACK KS in PARADISE has two variables, COL and SQ, and contains productions which know how to attack a particular square SQ for the side COL. (The ATTACK KS has more structure than this, but it is irrelevant to the present discussion.) With this KS, the system can use the abstract concept of attacking a square in its reasoning, in the expression of plans, and in the communication of discoveries from the tree search.

If one production knows about more than one abstract concept, it may be in more than one KS. When a concept in the data base (described below) has a corresponding KS (i.e., they have the same name), PARADISE will eventually treat that concept as a subgoal and use the corresponding KS to solve the subgoal in an attempt to produce a plan to realize that concept. (This may be done by creating other concepts.) There may also be concepts that have no corresponding KS and are not treated as subgoals. Such concepts are inspected by patterns attempting to match and by the searching routines. When a concept corresponds to a KS, it will sometimes be referred to as a goal or subgoal.

Concepts are posted in the data base by the actions in production rules. KSes then use the information in these concepts as subgoals to eventually produce a plan of action. When a new KS is executed, the patterns which matched in previous KSes can no longer be accessed. Thus concepts in the data base must contain all the information that will be needed to execute other KSs and to guide the search. Each KS expects a certain amount of information in a concept which it executes. (A concept is 'executed' when it is used as a goal by its corresponding KS.) At the very least, a KS expects an instantiation of its arguments (such as COL and SQ in the ATTACK KS). The ATTACK KS also expects some information on what the intentions of COL are once SQ is attacked. This information is needed so that the reason for wanting to attack SQ will not be destroyed or negated by an attempt to attack. Among other things, KSs usually expect information on how likely a goal is to succeed and on how threatening a goal is (this is useful in deciding if a sacrifice is warranted).

PARADISE stores information about concepts on a list of attribute-value pairs (or property list). A particular concept is expressed by giving its name, an instantiation of its arguments, and a list of attribute-value pairs, called an

attribute list. Frequently, instances of a concept with identical argument instantiations but different attribute lists are posted for different reasons by different productions. Since many productions do not care about the values in the attribute lists, it is imperative to treat all concepts with the same name and instantiation (though different attribute lists) as just one concept. Otherwise, many productions would needlessly be executed repeatedly on different versions of the same goal. Some productions do access the information in the attribute lists, and it is necessary to keep each attribute list distinct for these productions. For these reasons, all concepts in PARADISE with the same instantiation are combined into one concept (which is executed as a goal only once), but the attribute list of this one concept is replaced by a list of attribute lists, one for each reason the concept was suggested.

The productions can access this structure as one concept but can still access the necessary information about each attribute list. When a production accesses the values of attributes, it may not care which list the values come from, or it may require that the values for each attribute come from the same list, or it may want the 'best' (in some sense) value for each attribute whether the different values come from the same list or not, or it may want values for one attribute to come from lists which have some particular value for another attribute. This accessing of the information in a concept is so complex that it can be looked on as pattern matching in itself. Thus the productions in a KS can be looked upon as matching patterns in the given chess position and matching patterns in different concepts.

Concepts and their accessing are quite complex. Intuitively, the productions in PARADISE are not making deductions, per se, but coming up with ideas that may or may not be right. Thus there are no 'facts' that have certain deduced values that can be reasoned with. The complexity of later analysis requires that a production essentially put down 'why' it thought of an idea. In this way, other productions can decide if the idea still looks right in light of what is known at that time. Most production systems avoid this complexity because the firing of a production is a deduction which adds a new piece of knowledge to the system. This piece of knowledge is assumed right and the system is not prepared to later decide that it wasn't right to make that deduction.

To summarize, KSs provide abstract concepts that are at a higher level than the productions themselves, and form the language in which PARADISE thinks. Concepts and KSs are used in the static analysis reasoning process, to formulate plans for communicating knowledge down the tree, to quickly access relevant productions during execution of a plan, and to communicate discoveries made during the search.

5. Plans

The goal of the static analysis process in PARADISE is to produce plans. Plans seem to play an important role in the problem solving process of move selection for good human chess players. Alexander Kotov in his book *Think Like a Grandmaster* Kotov [7, p. 148] writes:

> ... it is better to follow out a plan consistently even if it isn't the best one than to play without a plan at all. The worst thing is to wander about aimlessly.

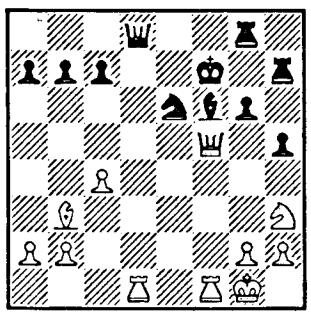
Most computer chess programs certainly do not follow Kotov's advice. In a knowledge based program, the cost of processing the knowledge should be offset by a significantly smaller branching factor in the tree search. Plans help reduce the cost of processing knowledge by immediately focusing the program's attention on the critical part of a new position created during the search, thus avoiding a complete re-analysis of this new position. Having plans also reduces the branching factor in the search by giving direction to the search. Consider, for example, that most programs may try the same poor move at every alternate depth in the tree, always re-discovering the same refutation. PARADISE can detect when a plan has been tried earlier along a line of play and avoid searching it again if nothing has changed to make it more attractive. Most programs also suffer because they intersperse moves which are irrelevant to each other and work towards different goals. PARADISE avoids this by following a single idea (its plan) down the tree.

Plans in PARADISE can be considered as special instances of concepts since each plan has a list of attribute lists. Instead of a list of instantiations for a set of variables (as in a concept), a plan has an expression in the *Plan-Language.* The Plan-Language expresses plans of action for the side about to move, called the offense. In general, the offense wants to have a reply ready for every defensive alternative. A plan cannot therefore be a linear sequence of goals or moves but must contain conditional branches depending on the opponent's reply. When the offense is on move, a specific move or goal is provided by the plan. When the defense is on move, a list of alternative sub-plans for the offense may be given. Each alternative begins with a template for matching the move made by the defense. Only alternatives whose template matches the move just made by the defense are tried in the search. PARADISE has the following templates which adequately describe defensive moves in terms of their effects on the purpose of the plan being executed (P and SQ are variables which will be instantiated in an actual plan to particular pieces and squares, respectively):

(P SQ) matches when the defense has moved the piece P to the square SQ.
(NIL SQ) matches when the defense has moved any piece to SQ.
(P NIL) matches when the defense has moved P (to any square).
(ANYBUT P) matches when the defense has moved some piece other than P.
NIL matches any defensive move.

The plan for an offensive move can be one of two things: a particular move or

a goal. A particular move is simply the name of a piece and a square. Such a plan causes PARADISE to immediately make that move with no analysis of the position other than checking that the move is legal. A goal is simply the name of a KS followed by an instantiation for each argument of the KS. When executing a goal, PARADISE will post a concept corresponding to the KS in the data base by using the given instantiations and the attribute lists of this plan as a whole. The KS will then be executed for this concept. Any plan produced by this execution will replace the goal in the original plan and this modified plan will be used. This process expands and ellaborates plans. If executing the KS does not produce a plan then the original plan has failed.



```
(((WN N5)
  (((BN N4) (SAFEMOVE WR Q7)
    (((BK NIL) (SAFECAPTURE WR BR))
     ((ANYBUT BK) (SAFECAPTURE WR BK))))
    ((BN N4) (CHECKMOVE WR Q7) (BK NIL) (SAFECAPTURE
                                        WR BQ) )))
((THREAT (PLUS (EXCHVAL WN N5) (FORK WR BK BR)))
                                        (LIKELY 0))
((THREAT (PLUS (EXCHVAL WN N5) (EXCH WR BQ)))
                                        (LIKELY 0)))
```

A plan produced by PARADISE

FIG. 3. White to move.

Fig. 3 shows a problem and one of the plans PARADISE's static analysis produces for it. ('WN' means white knight, 'N5' means the square N5, etc.) The internal representation of PARADISE has not been printed unambiguously since the term 'WR' does not specify which white rook. However, it should be obvious to which piece the names refer. The last two lines of the plan are attribute lists. The first five lines are the Plan-Language expression for the plan which can be read as follows:

> *Play N–N5. If black captures the knight with his knight, attempt to safely move the rook on Q1 to Q7. Then, if black moves his king anywhere try to safely capture the black rook on R7, and if black moves any piece other than his king, try to safely capture the king with the rook. A second alternative after black captures the knight is to attempt to safely move the rook on Q1 to Q7 with check. Then, if black moves his king anywhere try to safely capture the black queen with the rook.*

SAFEMOVE, SAFECAPTURE, and CHECKMOVE are all KSs in PARADISE. After playing N–N5 and N×N for black in the tree search, PARADISE will execute either the CHECKMOVE goal or the SAFEMOVE goal. Executing the SAFEMOVE goal executes the SAFEMOVE KS which knows about safely moving a piece to a square. This KS will see that R–Q7 is safe and produce this *as the continuation of the original plan, causing the system to play R–Q7 with no further analysis. If for some reason R–Q7 was not safe then the* SAFEMOVE KS will try to make Q7 safe for the rook (by posting a SAFE concept). If some plan is found, the original plan will be 'expanded' by replacing the SAFEMOVE goal with the newly found plan. In general, the newly found plan will contain (SAFEMOVE WR Q7) though it may come after a sacrificial decoy or some other tactic. If posting the SAFE concept does not produce a plan, then the SAFEMOVE goal has failed, and the alternative CHECKMOVE goal will be tried. If it also fails, a complete analysis of the position must be done. If black had answered N–N5 with a king move, it would not match the template (BN N4) and a complete analysis of the position would be undertaken without attempting to make Q7 safe for the rook.

By use of conditionals, a plan can handle a number of possible situations which may arise. The most important feature of the Plan-Language is that offensive plans are expressed in terms of the KSs. This has many advantages. Relevant productions are immediately and directly accessed when a new position is reached. The system has one set of abstract concepts it understands (the KSs) and does not need to use a different language for plans and static analysis. The system has been designed to make the writing of productions and thus the forming of KSs reasonable to do. Thus the range of plans that can be expressed is not fixed by the Plan-Language. By forming new KSs, the range of expressible plans may be increased without any new coding (in the search routines or anywhere else) to understand the new plans. This is important since the usefulness of the knowledge base is limited by its ability to communicate what it knows (e.g., through plans).

Given a number of plans to execute, the tree search must make decisions about which plan to search first, when to forsake one plan and try another, when to be satisfied with the results of a search, and other such things. To make such decisions, it must have information about what a plan expects to gain and why. Such information was available to the productions which matched to produce the plan and must be communicated in the attribute lists of the plan so as to provide many different types of access to this knowledge. Unlike most search-based chess programs, PARADISE must use its information about a plan at other nodes in the tree since it executes a plan without analyzing the newly created positions.

To use the information about plans in an effective manner, PARADISE divides a plan's effects into four different categories which are kept separately (i.e., their values are not combined in any way). These four categories are:

THREAT which describes what a plan threatens to actively win.

SAVE which describes counterthreats of the opponent a plan actively prevents,

LOSS which describes counterthreats of the opponent not defended against and functions the first piece to move in a plan will give up by abandoning its

current location (thus providing new threats for the opponent), and

LIKELY which describes the likelihood that a plan will succeed.

These four categories have emerged during the development of PARADISE. Experience seems to indicate the system must have at least these four dimensions along which to evaluate plans in order to adequately guide the tree search.

Since the values for these categories must evaluate correctly at different nodes in the tree, they cannot be integers. Instead they (except for LIKELY) are expressions in the Threat-Language [13] which can describe threats in a sophisticated manner. These expressions are evaluated in the context of the current position and may return different values for different positions. The value of LIKELY is an integer. This integer represents the number of unforced moves the defense has before the offense can accomplish its plan. The most likely plans to succeed have a LIKELY of zero (every move is forcing). Both attribute lists in Fig. 3 have a likelihood of zero.

6. Creating Plans

A detailed description of the static analysis process in PARADISE is given in [13]. The concepts and KSs used to produce the plan in Fig. 3 are briefly described here. The system begins by posting a THREAT concept, the THREAT KS having productions which look for threats. Two different productions in THREAT post MOVE concepts for moving the white rook to Q7. (One recognizes the skewer of the black king to the rook, and the other recognizes the simultaneous attack on the king and queen where any king move leaves the queen open to capture with check.) After executing the THREAT and ATTACK KSs, the system executes the MOVE KS on the 50 MOVE concepts which have been posted by productions. In addition to Q7, there are MOVE goals for moving the rook to QB7, K7, KB8, Q6, QB6, QN6, and QR6 as well as many goals for moving other white pieces. A production in the MOVE KS, that recognizes that the proposed move is unsafe, matches for the white rook to Q7 MOVE goal. This production posts a SAFE concept for making Q7 safe for the white rook. After the MOVE KS is finished, 7 SAFE goals have accumulated. While executing the SAFE KS for the white rook to Q7 goal, one production notices that the black knight blocks the white queen's protection of Q7. This production posts a DECOY goal in order to decoy the black knight so that the queen can support the white rook on Q7. The DECOY KS is executed on the 7 DECOY goals which have been produced, and one production posts a FORCE goal suggesting N–N5 as a move to decoy black's knight. It is up to the FORCE KS to decide if N–N5 is forcing enough to provoke a move by the black knight. There are 9 FORCE goals in all and the production which finds N–N5 forcing posts an INITIAL-PLAN concept which is exactly like the plan shown in Fig. 3.

After executing the CAPTURE KS, the INITIAL-PLAN KS is executed on the 5 INITIAL-PLAN goals which have been produced. (Most of the MOVE goals failed without leading to the posting of a plan.) This KS looks for functions given up by a plan or threats not defended against by a plan. Since the plan of Fig. 3 checks the opposing king, a production posts a FINALPLAN concept for this plan. While executing a different goal, the INITIAL-PLAN KS notices that the white queen is under attack and is not defended by the plan in question. This causes execution of the DEFENDOFFENSE KS which produces 6 DEFENDTHREAT goals to save the white queen. The DEFENDTHREAT KS in turn posts 6 INITIAL-PLAN concepts, and executing the INITIAL-PLAN KS a second time produces a total 10 FINALPLAN concepts for the position in Fig. 3. The attribute lists of these plans rate only two of them as being likely to win material. These two are the one in Fig. 3, and a plan starting with R–Q7 immediately. The plan in Fig. 3 is rated highest and is tried first. This static analysis took more than 24 seconds of cpu time on a DEC KL-10. The static analysis process is very expensive, and PARADISE uses its plan to avoid doing such analyses.

Each static analysis is done entirely by the production rules which can easily be modified to increase or make more precise the knowledge available to the system. Each concept (and plan) produced has a REASON attribute listing each production that matched in the process of suggesting the concept. (REASONs have been omitted in Fig. 3.) This provides a good explanation of exactly why PARADISE believes something and is helpful for quick debugging and easy modification. The attribute lists of each concept contain enough information so that the system essentially knows 'why' it is doing something. Thus wrong ideas can be discarded readily and combinations of ideas that are antithetical to each other can be avoided. Few ridiculous plans are suggested.

7. How Detailed Should Plans Be?

The system must quickly recognize when a plan is not working and abandon it before effort is wasted investigating poor lines. On the other hand, it is expensive to completely analyze a position, so the system wants to use its plan as long as possible. To achieve a balance, plans must adequately express the purpose they intend to achieve. Productions must carefully formulate their Plan-Language expressions at the right level of detail using appropriate goals and descriptions of defensive moves. When executing a plan, the system will retrieve a goal or move from the plan to apply to a new position. Care must be taken to ensure that this move or goal is reasonable in the new position. Plans may become unreasonable when they are too general or too specific. In either case, they have failed to express their purpose.

The example of Fig. 3 illustrates the issues involved. Consider the first three

ply of the plan as ((WN N5) (BN N4) (SAFEMOVE WR Q7)). Suppose this had been expressed as ((WN N5) NIL (WR Q7)). Playing R–Q7 after N × N is reasonable but if black moves his king instead, playing R–Q7 will lead the search down a blind alley. Thus, (WR Q7) is too specific and does not express white's purpose. This problem can be cured by a more specific description of black's reply (e.g., (BN N4) instead of NIL) since the template for black's move will not match if black moves his king. In actual fact, PARADISE does not know for sure that R–Q7 will be safe after N–N5 and N × N (although in this particular position it is). To avoid mistakenly playing R–Q7 when it is not safe, PARADISE uses a SAFEMOVE goal to more accurately express the purpose of the plan. For example, the plan could be ((WN N5) NIL (SAFEMOVE WR Q7)). If black answers N–N5 with N × N then the SAFEMOVE goal is reasonable and quickly produces R–Q7 as the move to play after verifying its safety. If black answers with a king move then the SAFEMOVE goal is not what white wants to do, but little is lost since R–Q7 is not safe and the line will be rejected without searching. However, if likely plans had existed for making Q7 safe, the search may still have been led astray. For this reason, PARADISE uses (BN N4) as the template in this plan. The purpose of N–N5 is to decoy the black knight to his KN4 and this template most accurately expresses this purpose.

Now let us consider the following more general plan: ((WN N5) NIL (SAFELY-ATTACK BK)). If black answers with N × N then the SAFELY-ATTACK KS should generate R–Q7 as a safe attack on the black king. If, instead, black moves his king then this KS should generate a check by the white queen or knight which would also be reasonable. Thus this plan produces reasonable moves for every black reply without ever causing a re-analysis of the new position. This would be a good plan if PARADISE knew (from its patterns) that it could get the black king in trouble after any reply. However, it only knows of the skewer of the black king to the rook, the threat of capturing the queen with check, and the fact that the white knight threatens the black king, rook, and knight from N5. It is accidental that the SAFELY-ATTACK goal works after black retreats his king. In the general case, such a goal would produce many worthless checks that would mislead the search. Thus this plan is too general to describe white's actual purpose.

It is very important to get the correct level of detail in a plan. The plan should handle as many replies as possible without causing a re-analysis, but it should avoid suggesting poor moves. The templates for describing defensive moves in the Plan-Language and the various KSs have been developed to allow PARADISE's plans to accurately express their purpose. The results have been quite satisfying: the productions in PARADISE now create plans which rarely suggest poor moves but which can still be used for as many ply as a human might use his original idea.

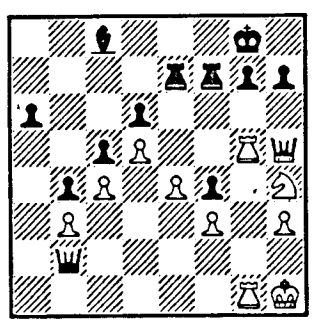
8. Using Plans to Guide the Search

PARADISE's tree searching algorithm is described in detail in [13]. Only a brief overview is given here in order to provide background for an example problem solution by the program. PARADISE's tree searching algorithm uses different strategies at the top level to show that one move is best. Once a strategy is selected, a best-first search is done. The value of each move is a range within which PARADISE thinks the 'true' value of the move lies. The program narrows these ranges by doing best-first searches until it can show that one move is best. By using the knowledge base to control the search and by using information discovered during previous searches, PARADISE produces enough cutoffs to force convergence of its search without a depth limit or other artificial effort limit.

Fig. 4 shows position 49 from [9]. PARADISE does a static analysis on this position which uses 18 seconds of cpu time and suggests two plans. Fig. 4 shows the plan suggested as best. To show how such plans are used to guide the search, PARADISE's search for this position is sketched below.

PARADISE begins by doing a best-first search of the above plan. Execution of this plan commences by playing Q × Pch and producing a new board position. The defense suggests both legal moves, K × Q and K–B1, but tries K × Q first. Since this move matches the (BK R2) template in the plan, PARADISE has ((CHECKMOVE WR R5) (BK NIL) (ATTACKP BK)) as its best plan at ply 3 of the search. Since it is in the middle of executing a plan, PARADISE does not look for better alternatives at this point. The best-first search only looks for better alternatives when a new plan is being selected.

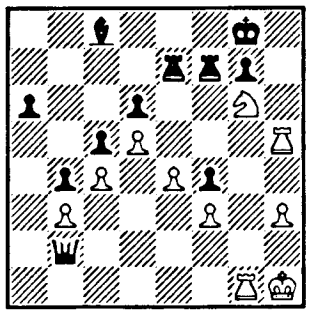
Execution of the CHECKMOVE KS produces ((WR R5) (BK NIL) (ATTACKP BK)) as a plan which causes R–R5ch to be played. Black plays his only legal-move at ply 4, K–N1, which matches the (BK NIL) template in the plan, and leaves (ATTACKP BK) as the current plan. Thus the original plan is still guiding the search at ply 5. Executing the ATTACKP KS posts many concepts, including MOVE and SAFE concepts, and ((WN N6) ((NIL (SAFEMOVE WR R8)) (NIL (SAFEMOVE WN K7))) is produced as the best



(PLY 1)
BESTPLAN:
(((WQ R7) (BK R2) (CHECKMOVE WR R5) (BK NIL)
(ATTACKP BK))
((HINT 1) (THREAT (PLUS (WIN BK) (EXCHVAL WQ R7)))
(LIKELY 1)))

A plan produced by PARADISE

FIG. 4. White to move.



(PLY 6)
INITIAL DEFENSIVE MOVES:
((BR R6) (BB Q2) (BB N2) (BR B4) (BR B3) (BR B1))

FIG. 5.

plan in this position. This replaces (ATTACKP BK) in the current plan, thus elaborating it. After playing N–N6, the position in Fig. 5 is reached.

Productions which know about defending against R–R8 suggest the 6 black moves in Fig. 5. Black plays B × P first in an effort to save the bishop from the ensuing skewer. At this point the plan branches. Since both branches begin with a null template, they both match any black move at ply 6. Thus PARADISE has two plans at ply 7: (SAFEMOVE WR R8) and (SAFEMOVE WN K7). Before executing a plan for the offense, PARADISE executes the QUIESCENCE KS which looks for obviously winning moves. (This was done at plys 3 and 5 also, but did not suggest any moves.) Here R–R8 is suggested by the QUIESCENCE KS which causes the (SAFEMOVE WR R8) plan to be executed immediately. PARADISE plays R–R8, finds that black is mated, and returns to ply 6 knowing that black's B × P leads to mate. All this has been accomplished without a static analysis; the original plan has guided the search.

At ply 6, black uses KSs to refute the mating line. This involves analysis of the information produced by the previous search. No new moves are found since all reasonable defenses have already been suggested. PARADISE has a causality facility (see [14]) which determines the possible effects a move might have on a line of play. Using information generated during the search of the mating line, the causality facility looks for effects a proposed move might have (such as blocking a square which a sliding piece moved over, vacating an escape square for the king, protecting a piece which was attacked, etc.). The causality facility recognizes that neither B–Q2 nor B–N2 can affect the mating line found for B × P so they are rejected without searching.

Black plays R–B4 next, the causality facility having recognized that this move opens a flight square for the black king. Again PARADISE has both (SAFEMOVE WR R8) and (SAFEMOVE WN K7) as plans at ply 7. Both SAFEMOVE goals would succeed, but R–R8 has a higher recommendation and is played first. Black plays his only legal move at ply 8, K–B2.

The original plan no longer provides suggestions at ply 9. PARADISE must now look for better alternatives or do an expensive static analysis to suggest a new plan. Before trying either of these, the program executes the QUIESCENCE KS in an attempt to find an obviously winning move. R–KB8 is suggested and PARADISE immediately plays this move without doing a static analysis. This is mate so PARADISE returns to ply 6 to look for other defenses.

Both R–B3 and R–B1 are tried and both are quickly refuted by playing R–R8 from the SAFEMOVE goal of the original plan, K–B2, and R–KB8 from the QUIESCENCE KS. The search then returns to ply 2 and tries K–B1 in answer to Q × Pch. The template in the original plan does not match K–B1 so there is no plan at ply 3. However, the QUIESCENCE KS quickly suggests Q–R8 and PARADISE returns from the search convinced that Q × Pch will mate. This result shows that Q × Pch is best, so no other best-first searches are initiated.

PARADISE's plans are not always so accurate but space prevents presentation of a longer search. The above analysis uses about 130 seconds of cpu time on a DEC KL-10. It goes to a depth of 9 ply while creating only 21 nodes in the tree. Because of the guidance provided by the original plan, no static analysis was performed except on the original position. By comparison, the TECH2 program (an improved version of TECH [6]) at a depth setting of 6 discovers that it can win material with Q × Pch although the horizon effect hides the mate from it. For this analysis, TECH2 uses 210 seconds of cpu time on a KL-10 and creates 439,459 nodes by making legal moves (as well as making 544,768 illegal moves which are retracted).

9. A Typical Medium-Sized Search

This section presents the actual protocol produced by PARADISE while solving problem 82 in [9]. (The program prints squares in algebraic notation to avoid confusion.) This is a typical protocol produced by PARADISE while solving a problem in which the program finds the best line immediately. Problem 82 in [9] was chosen so that comparisons can be made with the search tree produced

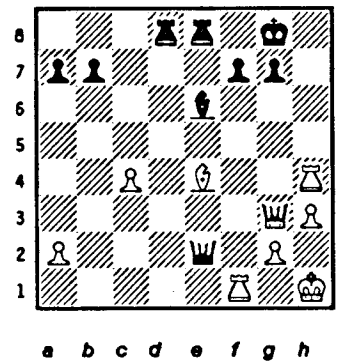


FIG. 5.1. White to move.

by the CAPS program which is given in [1]. Such a comparison is interesting since CAPS has more knowledge about tactics than any other program that plays a full game of chess.

PARADISE's initial static analysis on the above position uses about 20 seconds of cpu time, and produces 7 plans. The one beginning with (WB H7) has a LIKELY of 0 and is searched first. The other six plans have a LIKELY of 1 and begin with the following moves: (WQ A3) (WR H7) (WR H8) (WQ F2) (WQ F3) (WQ E1). The protocol which follows was printed by PARADISE as it analyzed this problem. Explanatory comments have been added in a smaller font.

In the search, the program refuses to search plans that are not likely to succeed when it has already achieved a significant success. This is done because it is not reasonable to invest large amounts of effort trying ideas which don't seem likely to succeed when a winning line has already been found. The validity of this cutoff rests on PARADISE's accurate calculation of LIKELY values (see [13]). This could produce errors, though not serious ones since the errors would be the selection of a winning move which was not the 'best' winning move. No such errors have been made in any position PARADISE has analyzed.

(TOPLEVEL (VALUE . 0))
PLY 1 (EXPECT . 320)
BESTPLAN: (WB H7) (BK F8) (WQ A3))
NEWBOARD: (E4 WB H7)

The value of the initial position is 0. (White is trying to achieve a positive score.) The best plan specifies moving the WB to H7 and if black replies by moving his king to F8, the WQ–A3 is played. The reasoning behind the suggestion of this plan is fairly specific. PARADISE knows that after B–H7 black must move his king either to H8 where white has a discovered check, or to F8 where white can deliver a second check to which black's king cannot move in reply. The expectation of this plan (after 'EXPECT'), which is calculated from the attribute lists of the plan, is 320 (90 is the value of a queen, so 320 threatens mate). Wherever the word 'NEWBOARD' occurs, PARADISE constructs a new board position by playing a legal move.

(PLY: 2) DEFMOVES: ((BK F8) (BK H8))
NEWBOARD: (G8 BK F8)

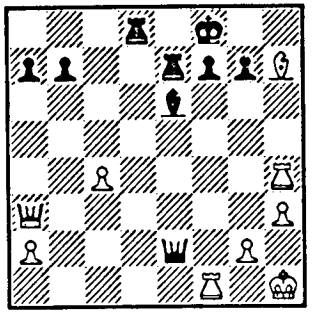
The word 'DEFMOVES' labels the list of moves under consideration by the defensive search. (Each move is the first part of a plan under consideration.)

TRY CURRENT PLAN
PLY 3 (VALUE . 0) BESTPLAN: ((WQ A3))
NEWBOARD: (G3 WQ A3)

The phrase 'TRY CURRENT PLAN' means that the program is obtaining the next move by executing a plan that was inherited from an earlier analysis. In this example the current plan specifies an actual move rather than a goal, so the move is made immediately without calculating the primitives in this position. Applying the evaluation function to this position yields 0 (given after the word 'VALUE').

(PLY: 4) DEFMOVES: ((BR E7))
NEWBOARD: (E8 BR E7)

PARADISE knows (BR D6) will not help; CAPS searches it after refuting (BR E7).



NULL PLAN (VALUE . 0) (STATIC-ANALYSIS 20.4 SECONDS) (4 PLANS)
PLY 5 BESTPLAN: (((WB D3) ((NIL (CHECKMOVE WR H8)) (NIL (SAFECAPTURE WB BQ))
(NIL (SAFEMOVE WR H8) (((BK NIL) (SAFECAPTURE WR BR))
((ANYBUT BK) (SAFECAPTURE WR BK)))))))
(NEWEXPECT . 90)
NEWBOARD: (H7 WB D3)

The current plan is finished so the system does a static analysis which takes 20.4 seconds of cpu time and produces 4 plans. The best one begins with (WB D3) and continues with either a rook check on H8 threatening mate, the capture of the black queen by the bishop, or the rook move to H8 followed by a skewer of the black king to the black rook. Once again the static analysis accurately recommends the winning plan. This is an improvement over the performance of CAPS which finally suggested B–D3 as a defensive move since it protected the white pawn and white rook which are both en prise. The expectation is now 90, having been recalculated from the new current plan which only expects to win the black queen.

(PLY: 6) DEFMOVES: ((BR D3) (BR D3) (BQ D3) (BQ F1) (BP G5) (BQ D3) (BK G8))
NEWBOARD: (D8 BR D3)

TRY OBVIOUSLY WINNING MOVE
PLY 7 (VALUE . -33) BESTPLAN: ((WR H8))
(NEWEXPECT . 607)
NEWBOARD: (H4 WR H8)
(EXIT OFFENSE (VALUE 1300 . 1300))

The system finds R–H8 as an obviously winning move. It knows the move will mate (although it plays it to make sure), so it doesn't check the current plan. If the system hadn't been sure of the mate, it would have executed the current plan after noticing that R–H8 duplicates the

(CHECKMOVE WR H8) goal in the current plan. A range is returned as the value, but both the top and bottom of the range are 1300 (the value for mate) since the value has been exactly determined.

(PLY 6) REFUTE: ((WR H8))
(PLY: 6) DEFMOVES: ((BQ D3) (BQ F1) (BP G5) (BQ D3) (BK G8))
CAUSALITY: (BQ D3) LINE: (D8 BR D3) NO
NEWBOARD: (E2 BQ F1)

PARADISE backs up to ply 6. A counter-causal analysis tries to refute the move R–H8 by white but no new moves are suggested. The causality facility compares the proposed Q–D3 move to the tree produced for the R–D3 move and determines that Q–D3 cannot help. Q–D3 is therefore rejected without searching and Q–F1 is played (the causality facility approves it). The causality facility makes use of considerable information returned by the searching process. CAPS, which also has a causality facility, searches both R–D3 and Q–D3 in this position. In fact, more than half the moves rejected by causality in the remainder of this protocol are searched by CAPS.

OBVIOUSLY WINNING MOVE DUPLICATED
TRY CURRENT PLAN
PLY 7 (VALUE . −50) BESTPLAN: ((WB F1))
NEWBOARD: (D3 WB F1)

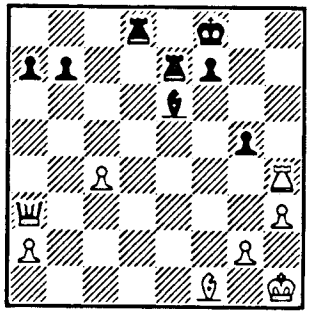
B–F1 is suggested as a winning move, but it duplicates the SAFECAPTURE goal in the current plan (and is not a sure mate) so the current plan is tried.

(PLY: 8) DEFMOVES: ((BP D1))
NEWBOARD: (D8 BR D1)

(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (H4 WR H8)
(QUIESCENCE VALUE (1300 . 1300))

The current plan is finished so the offense tries a quiescence search to see if the value holds up. A value of 1300 is returned so the search backs up with success for white.

(PLY 8) REFUTE: ((WR H8))
(PLY: 8) DEFMOVES: ((BP G5) (BR D2) (BR D7) (BP G6) (BP F5) (BP F6) (BK E8) (BK G8))
NEWBOARD: (G7 BP G5)



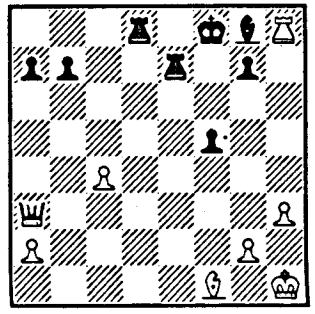
(VALUE .49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (H4 WR H8)
DEFENDING MOVE SELECTED
(PLY: 10) NEWBOARD: (F8 BK G7)
(PLY: 11) NEWBOARD: (H8 WR D8)
DEFENDING MOVE SELECTED
(PLY: 12) NEWBOARD: (E7 BR D7)
(PLY: 13) NEWBOARD: (D8 WR D7)
(PLY: 14) NEWBOARD: (E6 BB D7)
(PLY: 15) NEWBOARD: (A3 WQ A7)
DEFENDING MOVE SELECTED
(PLY: 16) NEWBOARD: (D7 BB C6)
(QUIESCENCE VALUE (109 . 109))
(EXIT OFFENSE (VALUE 109 . 109))

A quiescence search to a depth of 16 shows that P–G5 fails for black. The quiescence search knows that white is ahead and that the white rook on H4 is not in danger, yet it plays aggressive moves (at plys 9 through 15) instead of being satisfied. This seems to generate nodes unnecessarily. PARADISE does this because it is a cheap way to get better results. The quiescence search is very inexpensive compared to the regular search. The system thinks it has enough information, yet it may be necessary to search this line again if the 'true' value is not found. Since PARADISE sees an inexpensive way (quiescence searching) to improve the result, it risks generating a few unnecessary (though inexpensive) nodes in order to avoid a possible re-search later.

(PLY 8) REFUTE: ((WR H8))
(PLY: 8) DEFMOVES: ((BR D2) (BR D7) (BP G6) (BP F5) (BP F6) (BK E8) (BK G8))
CAUSALITY: (BR D2) LINE: (D8 BR D1) NO
CAUSALITY: (BR D7) LINE: (D8 BR D1) NO
CAUSALITY: (BP G6) LINE: (G7 BP G5) NO
NEWBOARD: (F7 BP F5)

Black tries other moves at ply 8. Causality rejects R–D2 and R–D7 on the basis of the tree generated for R–D1, and it rejects P–G6 using the P–G5 tree.

(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (H4 WR H8)
DEFENDING MOVE SELECTED
(PLY: 10) NEWBOARD: (E6 BB G8)



(PLY: 11) NEWBOARD: (A3 WQ A7)
(QUIESCENCE VALUE (59 . 59))
(EXIT OFFENSE (VALUE 59 . 90))

Again the quiescence search confirms the offensive success. This time the value of 59 does not meet the expectation so the offensive search returns a range of 59 to 90 since the expectation of 90 may have been achieved if a static analysis had been done. PARADISE notes in the tree that searching here again may improve the value from 59 to 90.

(PLY 8) REFUTE: ((WR H8))
(PLY: 8) DEFMOVES: ((BP F6) (BK E8) (BK G8))
CAUSALITY: (BP F6) LINE: (F7 BP F5) NO
NEWBOARD: (F8 BK E8)

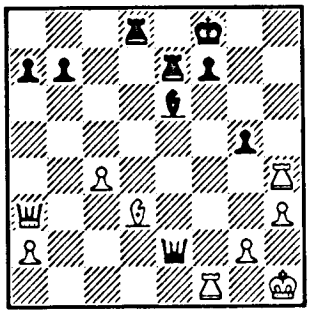
(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (A3 WQ A7)
(QUIESCENCE VALUE (59 . 59))
(EXIT OFFENSE (VALUE 59 . 90))

(PLY 8) REFUTE: ((WQ A7))
(PLY: 8) DEFMOVES: ((BK G8) (BR A8) (BP A6) (BP A5) (BP B6))
CAUSALITY: (BK G8) LINE: (F8 BK E8) NO
CAUSALITY: (BR A8) LINE: (D8 BR D1) NO
CAUSALITY: (BP A6) LINE: (D8 BR D1) NO
CAUSALITY: (BP A5) LINE: (D8 BR D1) NO
CAUSALITY: (BP B6) LINE: (F8 BK E8) NO
DEFMOVES: ( )

PLY 7 BESTPLAN: ((CHECKMOVE WR H8))
TERMINATE: PLAN SUCCEEDED
(EXIT OFFENSE (VALUE 59 . 90))

The offensive search terminates since a significant gain has been achieved. PARADISE again notes in the tree that searching other plans here may improve the value. The range of 59 to 90 is returned.

(PLY 6) REFUTE: ((WR H8) (WR H8) (WR H8) (WB F1))
(PLY: 6) DEFMOVES: ((BP G5) (BK G8))
NEWBOARD: (G7 BP G5)



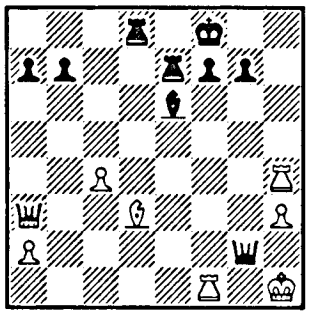
OBVIOUSLY WINNING MOVE DUPLICATED
TRY CURRENT PLAN
PLY 7 (VALUE . 0) BESTPLAN: ((WB E2))
NEWBOARD: (D3 WB E2)

(PLY: 8) DEFMOVES: (BP H4))
NEWBOARD: (G5 BP H4)

(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (A3 WQ A7)
(QUIESCENCE VALUE (59 . 59))
(EXIT OFFENSE (VALUE 59 . 90))

(PLY: 8) DEFMOVES: ( )
PLY 7 BESTPLAN: ((CHECKMOVE WR H8))
TERMINATE: ALPHA BETA

(PLY 6) REFUTE: ((WB E2))
(PLY: 6) DEFMOVES: ((BQ E3) (BQ D2) (BQ E5) (BQ G2) (BQ A2) (BK G8))
CAUSALITY: (BQ E3) LINE: (D8 BR D3) NO
CAUSALITY: (BQ D2) LINE: (D8 BR D3) NO
CAUSALITY: (BQ E5) LINE: (D8 BR D3) NO
NEWBOARD: (E2 BQ G2)



TRY OBVIOUSLY WINNING MOVE
PLY 7 (VALUE . −10) BESTPLAN: ((WK G2))
NEWBOARD: (H1 WK G2)
(PLY: 8) DEFMOVES: ( )

The defensive search now does a null move analysis (see Wilkins [13]). It has no idea what to do so it lets the offense make two moves in a row. However, the offense calls the quiescence search which knows it is the defense's move. It decides the defense can escape from the threat of R–H8 so it calls the position quiescent.

(VALUE . 89) TRY QUIESCENCE SEARCH
assumed escape from (WR H8)
(QUIESCENCE VALUE (89 . 89))
(EXIT OFFENSE (VALUE 89 . 89))

(PLY: 8) DEFMOVES: ( )
(PLY: 6) REFUTE: ((WK G2))
(PLY: 6) DEFMOVES: ((BQ A2) (BK G8))
CAUSALITY: (BQ A2) LINE: (D8 BR D3) NO
CAUSALITY: (BK G8) LINE: (G7 BP G5) NO
(PLY: 6) DEFMOVES: ( )

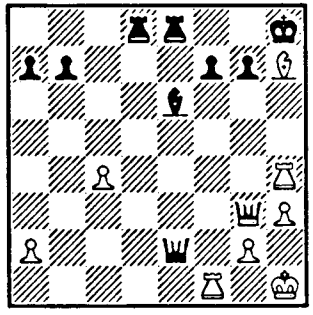
PLY 5 BESTPLAN: ((WQ A5) NIL (CHECKMOVE WQ D8))
no unlikelys after success: quit
(EXIT OFFENSE (VALUE 59 . 89))

The search backs up to ply 5 and tries the next plan suggested by the static analysis. This plan has a LIKELY of 1 and the result already achieved is so successful that PARADISE rejects the unlikely plan without searching.

(PLY 4) REFUTE: ((WR H8) (WB D3))
(PLY: 4) DEFMOVES: ( )
PLY 3 LAST PLAN
(EXIT OFFENSE (VALUE 59 . 320))

The search backs up to ply 3. There are no more plans to execute here, but a static analysis has not yet been done so the program makes a note of this. The top of the value range is changed to 320 to reflect the possibility of achieving the original expectation by doing a static analysis to find plans.

(PLY 2) REFUTE: ((WQ A3))
(PLY: 2) DEFMOVES: ((BK H8))
NEWBOARD: (G8 BK H8)



NULL PLAN (VALUE . 0) (STATIC-ANALYSIS 14.2 SECONDS) (2 PLANS)
PLY 3 BESTPLAN: ((WB D3))
(NEWEXPECT . 320)
NEWBOARD: (H7 WB D3)

The defense tries K–H8 at ply 2 and the original plan does not match this move so a static analysis is undertaken. Again PARADISE finds the right idea immediately. To compare, CAPS did not have the necessary knowledge and tried 3 other moves first, generating 80 nodes to disprove them, before stumbling onto B–D3.

(PLY: 4) DEFMOVES: ((BK G8))
NEWBOARD: (H8 BK G8)

(VALUE . 0) TRY OBVIOUSLY WINNING MOVE
PLY 5 BESTPLAN: ((WB E2))
(NEWEXPECT . 90)
NEWBOARD: (D3 WB E2)

(PLY: 6) DEFMOVES: ( )
(VALUE . 99) TRY QUIESCENCE SEARCH
(QUIESCENCE VALUE (99 . 99))
(EXIT OFFENSE (VALUE 99 . 99))

(PLY: 6) DEFMOVES: ( ) PLY 5 LAST PLAN
(EXIT OFFENSE (VALUE 99 . 320))
(PLY 4) REFUTE: ((WB E2)) (PLY: 4) DEFMOVES: ( )

PLY 3 BESTPLAN: ((WB E4))
(NEWEXPECT . 10)
TERMINATE: FORWARD PRUNE
(EXIT OFFENSE (VALUE 99 . 320))

The search backs up to ply 3 and the offense tries B–E4 which has been suggested by the static analysis. Its expectation is 10 and B–D3 has already achieved 99 so a forward prune occurs.

(PLY 2) REFUTE: ((WB D3))
(PLY: 2) DEFMOVES: ( )

PLY 1 BESTPLAN: ((WQ A3) NIL (CHECKMOVE WB H7))
no unlikelys after success: quit

The search returns to the top level to try the next plan but it is terminated because it has a LIKELY of 1 and the search is over. CAPS invests much effort at this point searching other offensive moves.

BEST MOVE: (E4 WB H7) VALUE: (59 . 320)
PRINCIPAL VARIATION:
1 (E4 WB H7) (G8 BK F8)
2 (G3 WQ A3) (E8 BR E7)

3 (H7 WB D3) (E2 BQ F1)
4 (D3 WB F1) (F7 BP F5)
5 (H4 WR H8) (E6 BB G8)
6 (A3 WQ A7)

TOTAL TIME: 297 SECONDS
NODES CREATED: 36 (22 REGULAR, 14 QUIESCENCE)
STATIC ANALYSES: 2

WHERE TIME WAS SPENT:
49% calculating primitives
33% quiescence searching (overlaps primitive calculations for 14 nodes)
11% static analysis
9% defense determining initial move
3% disk IO
0% causality facility, evaluation function, creating board positions

Many things should be noticed in this example. The plans do an excellent job of guiding the search: only two static analyses are done in the entire search. The plans are so accurately specified by the analysis that they never once lead the search off the correct line. White's play in the search is error-free. The causality facility makes good use of information returned from the search. Many black moves which would otherwise have been searched are eliminated in this manner. The range values accurately express the potential of each node so that the system does not have to waste effort determining the 'true' values. The best first search strategy plays a minor role in this example.

On this problem, CAPS generated a tree of 489 nodes in 115 seconds to obtain a principal variation of B–R7ch, K–B1, Q–R3ch, R–K2, B–Q3, Q × Rch, B × Q, R–Q8. This is slightly inaccurate since there is a mate in one for white at the end of this variation which CAPS's quiescence analysis did not recognize, but it is clear that CAPS understands the problem and its solution. PARADISE generates only 36 nodes to CAPS's 489 for the following three reasons (primarily):

—PARADISE has more knowledge available during static analysis and can accurately analyze a position and produce good plans. CAPS generates many nodes by not playing the correct offensive move first on some occasions.

—PARADISE returns more useful information from its search and can therefore use its causality facility to eliminate moves that CAPS searches.

—PARADISE has a best-first search strategy while CAPS is committed to finding the 'true' value (within alpha-beta) of each node it searches. This enables PARADISE to terminate as soon as some information is discovered without having to look for better alternatives.

This comparison shows the advances PARADISE has made in the use of knowledge. It is not meant to belittle CAPS which pioneered some of the techniques basic to this approach. In fact, CAPS is the only program with which a comparison is appropriate. For example, on this problem CHESS 4.4 (running on a CDC-6400) produces a tree with 30,246 nodes in 95 seconds of cpu time without finding the solution (which is too deep for it). Looking at the details of such a tree would not be helpful.

10. Measuring PARADISE's performance

To aid in evaluating performance, PARADISE was tested on positions in the book *Win At Chess* [9]. This book contains tactically sharp positions from master games which are representative of tactical problems of reasonable difficulty. PARADISE's knowledge base was developed by writing productions which would enable the program to solve, in a reasonable manner, fifteen chosen positions from among the first 100 in *Win At Chess*. The 85 positions not chosen were not considered during this development. This development process produced one version of the program, called PARADISE-0. Six positions were then picked at random from the remaining 85 and accurate records were kept on the work involved in getting the program to solve these reasonably. The version of the program which solves all 21 positions is called PARADISE.

PARADISE's performance on a problem is classified into one of three categories:

(1) problem solved as is (possibly with a minor bug fix),

(2) problem not solvable without a change to the program or a major change to the knowledge base, or

(3) problem solvable with a small addition to the knowledge base.

Category 3 helps measure the modifiability of the knowledge base. It is meant to include solutions which require no changes whatsoever to the program and only a small addition to the knowledge base. Small means that it takes less than 20 minutes total of human time to identify the problem and write or modify one production which will enable PARADISE, with no other changes, to solve the problem in a reasonable way (i.e., no ad hoc solutions).

Of the six positions chosen at random, two were in category 1, three were in category 3, and one was in category 2. The latter one inspired the only program changes between PARADISE-0 and PARADISE. These results speak well for the modifiability of the knowledge base, but shed little light on the generality of the program. To better test generality, PARADISE has been tested on the first 100 positions. These positions are divided into 5 groups of 20 and Reinfeld claims an increase in difficulty with increase in group number. (Eight end-game positions were eliminated, so the groups actually have 18, 19, 18, 20, and 17 positions.) PARADISE is considered to solve only the problems in category 1, while PARADISE-2 solves both category 1 and 3 problems. PARADISE would become PARADISE-2 simply by leaving in the productions written to solve problems in category 3.

Since the 21 developmental problems had not all been solved by one version

of the program, these were tried first. Adding new knowledge during program development would (hopefully) not adversely affect performance on earlier problems, but this had to be confirmed. For example, productions added to the knowledge base to solve the last 5 developmental problems might produce so many suggestions in the first 5 problems that the search would become untractable in those problems. The testing proved that the new knowledge had no adverse effects. All 21 problems fell into category 1; thus the same version of the program solves them all. In every case, the analysis is either the same as or sharper than that produced by developmental versions of the program. This result provides strong evidence that the knowledge base is easily modifiable. If productions are written carefully and intelligently (a skill the author developed while writing productions for the developmental set), they do not appear to adversely affect the system's performance on positions unrelated to the new productions. This is an essential quality for a modifiable knowledge base.

Table 1 shows what percentage of these problems can be solved by PARADISE, PARADISE-2, CAPS (Berliner [1]), TECH (Gillogly [6]), CHESS 4.4 (Slate [12]) running on a CDC 6400, and a human player rated as class A (Berliner [1]). PARADISE was limited to forty-five minutes of cpu time per problem while the other programs were limited to five minutes.

PARADISE already exhibits more generality in this domain than programs like TECH and CAPS. PARADISE-2 outperforms all the programs and the human. This shows that these problems do not push the limits of the expressibility of the production language nor the ability of the program to control the tree search. PARADISE does well on Group III because seven of those twenty problems are in the twenty-one problems on which PARADISE was developed.

There are twenty problems solved by PARADISE-2 but not by PARADISE, but only thirteen productions were written to solve them. In two instances, an already existing production was modified. In five instances the same production solved two different category three problems. This is a strong indication that the productions being written are fairly general and not tailored to the specific problem. These results indicate that the generality of PARADISE is reasonable and that the modifiability of the knowledge base is excellent. There are three problems not solved by PARADISE-2 which are discussed in the next section.

TABLE 1. Percentage of problems solved by various chess players

	PARADISE	PARADISE-2	CAPS	TECH	CHESS 4.4	Class A human
Group I	78%	100%	67%	78%	94%	89%
Group II	68%	95%	74%	84%	95%	95%
Group III	94%	100%	61%	61%	78%	94%
Group IV	70%	95%	50%	40%	70%	80%
Group V	65%	94%	41%	47%	76%	53%
All 92	75%	97%	59%	61%	83%	83%

TABLE 2. Comparison of average tree size on first 100 problems (for solved problems)

	PARADISE-2	CAPS	CHESS 4.4
Group I	19.8	167.8	24,907.3
Group II	28.7	226.4	34,726.0
Group III	35.0	206.1	24,200.1
Group IV	58.4	453.6	31,538.1
Group V	48.4	285.3	25,917.5
All 100	38.1	260.6	28,496.8

PARADISE uses more cpu time and produces trees with considerably fewer nodes than the other programs. The example search in Section 8 is a problem from Group III, and tree sizes of 21 nodes for PARADISE and 439, 459 for TECH-2 are fairly typical. Table 2 compares the average tree size (in number of nodes) for PARADISE-2, CAPS and CHESS 4.4 on the problems they solved in these 100.

PARADISE attempts to use knowledge whenever possible in order to produce trees of the same order of magnitude as those produced by human masters. Table 2 shows that this has been accomplished for the most part. CAPS uses a lot of knowledge but invests only about one-fifth of a second per node calculating. CAPS relies on the search to correct mistakes made by inadequate knowledge and this results in trees one order of magnitude larger than those produced by PARADISE. CHESS 4.4 has little chess knowledge and relies almost entirely on search to discriminate between moves. It generates trees that are three orders of magnitude larger than those generated by PARADISE. The two knowledge-oriented programs (CAPS and PARADISE) grow larger trees for the deeper combinations (Groups IV and V), just as most humans would. CHESS 4.4's tree size seems unrelated to the depth of the combination. CHESS 4.4 is of course the best chess player of these three programs.

Table 3 shows where PARADISE invests its resources. The statistics are

TABLE 3. Resource investment in PARADISE (89 solved problems)

	Mean	Highest	Lowest	Standard Deviation
CPU time for whole problem:	332.9	1958	19	396.6
Nodes created during search:	38.06	215	3	41.9
% CPU time calculating primitives:	52.85	74.3	34.1	8.47
CPU time per static analysis:	12.22	26.5	2.2	5.14
Number of static analyses per problem:	3.73	35	0	5.96
% CPU time doing static analysis:	11.61	33.7	0.0	10.04
% nodes requiring static analysis:	8.95	33.3	0.0	8.07
% CPU time executing knowledge base:	91.6	97.7	81.5	2.92

compiled over the 89 problems which PARADISE-2 *solved. They give some idea of the size of tree* PARADISE *grows, the amount of time it spends, and where this time is spent. (CPU time is in seconds on a KL-10 processor.)*

Table 3 shows that calculating primitives is PARADISE's most computationally significant activity. The fact that less than 9% of the nodes generated had static analyses done on them shows that the plans do a good job of guiding the search. Because of this, static analyses are not a major overall expense even though an average one takes more than twelve seconds of cpu time. The different cpu time limit given to PARADISE in Table 1 is due to the availability of statistics, but is not as unfair as it seems. As Table 3 shows, PARADISE-2 uses an average of only 5.5 minutes of cpu time on the 89 problems it solves. Also, PARADISE could probably be speeded up by a factor of 2 or more with more efficient production matching. As Table 3 shows, the program spends more than 91% of its time in the inefficient production interpreter.

11. How PARADISE Goes Wrong

PARADISE-2 was not able to solve 3 of the 92 problems after small additions to the knowledge base. A failure by PARADISE can be classified in one of the three following categories:

(1) The best plan is never suggested. PARADISE tries every plan that has been suggested and then quits without having achieved an acceptable result since it has run out of ideas. This can be cured by filling a hole in the program's knowledge so that the best plan will be suggested.

(2) The search becomes unbounded. If a position is rich in possible attacks and defenses, with most combinations running many ply before a quiescent position is reached, then PARADISE may use an unreasonable amount of time searching. During testing, PARADISE was limited to 45 minutes of cpu time per problem.

(3) A mistake is made in the analysis. For example, PARADISE may not solve a problem (even though recommending the best move) because the causality facility has eliminated a good defensive move which should have been searched, or because the quiescence analysis has made a large error (small errors must be tolerated).

Of the twenty problems solved by PARADISE-2 but not by PARADISE, 19 fell into category 1 initially. One fell into category 3 because of a mistaken quiescence analysis, but an added production fixed the problem. There were no failures caused by the search becoming unbounded. The 3 problems which PARADISE-2 could not solve are shown in Fig. 6, and are examined in detail below.

In problem 31, white plays P–Q6 and wins the black bishop which must move to save the black rook. If Q × R is played immediately, black replies B × Pch and wins white's queen. (White still has a won position after this, but it is an

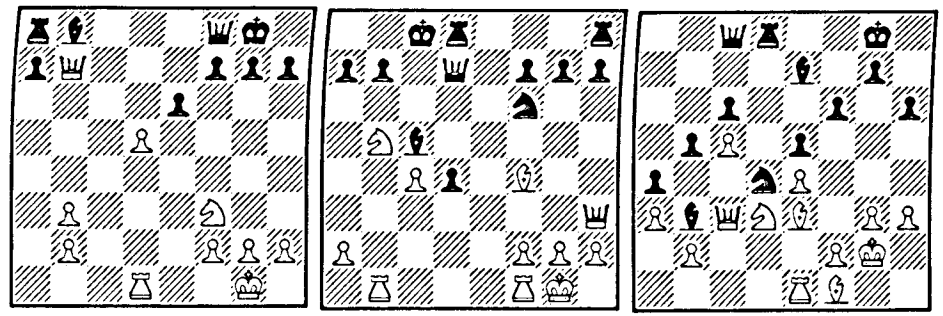


FIG. 6. Problems not solved by PARADISE-2.

involved pawn ending which goes well beyond the depth to which any current program can reasonably search.) PARADISE-2 could easily solve this problem by including a production which tries to block a checking move by a piece blocking a support of an en prise piece. Such a production could be expressed in only a few lines in the production-language. However, this is unacceptable because it is an ad hoc solution. The 'correct' way to find P–Q6 is by analyzing the refutation of Q × R. This move is suggested by PARADISE's counter-causal analysis but is not searched because offensive counter-causal moves are tried only if they are caputures (see [13]). Changing this restriction would involve a program change, so PARADISE-2 cannot solve this problem. This problem would be classified as a category 3 failure.

Problem 71 is a category 1 failure. The initial static analysis does not suggest N × RPch which wins for white by enabling the white queen to move to QR3 and then up the queen rook file. The static analysis has the goal of moving the white queen to QR3 and then up the queen rook file, so a fairly simple production in the MOVE KS suggests N × RP at the top level. With this new production PARADISE-2 at first got the correct answer, but a bug in one of the productions in the DEFENDMOVE KS prevents some good defensive moves from being tried at ply 4. When this defensive production is fixed, the search becomes unbounded (i.e., uses more than 45 minutes of cpu time) because of the many defensive and offensive possibilities. This is the only example of a category 2 failure during testing of the program.

Position 91 is also a category 1 failure. Black can win a pawn and a much superior position by playing B–K3. Black threatens mate with his queen and knight and white must play B × N to avoid it, allowing P × B, 2. Q–any P × Pch winning a pawn. Whenever PARADISE suggests mating sequences, it first expends a large amount of effort doing a sensitive analysis of the upcoming possibilities. Problem 91 must be understood on a similar level of specificity, since PARADISE

uses this specific analysis to control the search. However, such a specific production would require more than thirty minutes of human effort to design and implement so PARADISE-2 was not given credit for this problem.

12. Comparison to Plans in Robot Problem Solving

Plans are used in many domains of AI research, especially robot problem solving. ABSTRIPS (Sacerdoti [10]), NOAH (Sacerdoti [11]), and BUILD (Fahlman [5]) are examples of planning systems in robot problem solving environments. Plans in these systems are very different from those in chess so a general description of these differences is more appropriate than a detailed comparison. Robot planning is frequently done in abstracted spaces which have been defined. These abstracted spaces omit details such as whether a door is open or not. There are a small number of such details, and the problem solver has some specific knowledge for each such detail which it can apply when required. For example, the problem solver has a procedure for opening a door should the door be closed when the robot wants to go through it. Such an approach is much more difficult in chess, at least with mankind's current understanding of the game. It is not clear how to define abstracted spaces. Small details are very important in chess and cannot be readily ignored, even for determining the first step of a plan. For example, a chess planner should not ignore the fact that a piece is pinned on the assumption that it can be unpinned when the plan calls for it to move. Thus plans in PARADISE are all at the same, very detailed level.

In the robot planning systems, the effects of an action are well defined. The program can easily and quickly determine the exact state of the world after it has moved a blue block. In chess, moves (actions) may subtly affect everything on the board. For example, a piece that was safe may no longer be (even though the move made has no direct effect on the piece), and the system may need to make an expensive calculation to determine this. Chess plans cannot make many assumptions about the state of the world in the future, but must describe the expected features of new states in the plan itself and then test for these features while executing the plan. Consider the monkey and bananas problem. If the experimenter is sinister, he could connect the bananas to the box by pulleys and rope, invisible to the monkey, so that when the box is pulled under the bananas, the bananas are pulled up so they cannot be reached from atop the box. In a sense, the robot planners assume that no such crazy side effects will happen, while a chess plan must prepare for such things. This manifests itself in PARADISE in at least two ways. First, plans are not composed of simple actions but are goals which may require the knowledge in many productions to be interpreted. This postponement of evaluation permits checking of many complex features in the current world state. Second, PARADISE has been designed to make it easy to produce new concepts (by writing productions and forming KSs) for expressing plans whenever they are needed to handle new 'crazy side effects.'

In most robot planning systems, tests for having achieved the goal or the preconditions of an action are trivial and give well defined answers. In chess, there may be only very subtle differences between a position where a particular action is good and positions where the same action is wrong. It is also hard to know when a goal has been achieved, since there is always a chance of obtaining a larger advantage (except when the opponent has been checkmated). Thus in chess it is necessary for the plan to provide a considerable amount of information to help in the making of these decisions. This is done in PARADISE through the attribute lists in plans.

The number of things PARADISE can consider doing at any point in a plan is about an order of magnitude larger than the number of things most robot problem solvers usually contend with. There are an average of about 38 legal moves in a chess position (see [3]). A robot usually has a much smaller number of possible actions which cannot be easily eliminated (for example, going to one of a few rooms or picking up one of a few objects). When planning farther in the future than just the first action, the chess planner has many more choices than the 38 legal moves. The inability to make assumptions about future states often prevents mentioning actual moves in a plan. Instead a description of the intent is needed, and the number of such descriptions is much larger than the number of legal moves that might be made. A robot usually has a much smaller number of possible actions, so the two types of plans explode at very different rates.

To summarize, the robot planners have the flavor of establishing a sequence from a small number of well-understood operations until the correct order is found, while chess planners have more a flavor of 'creating' the correct plan from the many possibilities. For this reason, a system which produces good chess plans needs a large amount of knowledge and non-trivial reasoning processes to produce plans.

Probably the most important idea PARADISE has for the robot problem solvers is that a plan may be viewed as a way to control what parts of a large knowledge base will be used to analyze each situation during plan execution. Plans in PARADISE can be viewed as telling the system what knowledge to use in analysis, thus defining a perspective for the system to use in new situations. Some amount of analysis from this new perspective is almost always done before the plan can continue executing. In most robot planning systems plans are more accurately viewed as telling the system what action to take. PARADISE tries to delay execution in order to bring more knowledge to bear.

13. Comparison to Plans in Chess

Pitrat's program [8], which solves chess combinations, is the most important

example of the use of plans in chess. The language Pitrat uses for expressing plans has four statements with the move statement and modification statement being the most important. The move statement specifies that the piece on one particular square should move to another particular square. It may also be specified that the move must be a capture. The modification statement describes a modification to be made to a particular square. This can be one of four things: removal of a friend, removal of an enemy, moving a friend to the square, or getting an enemy to move to the square.

Plans in PARADISE provide much more flexibility in expression that Pitrat's move statement. In PARADISE, types of moves other than captures can be specified (e.g., safe moves and safe capture moves). Also, particular squares do not need to be named in PARADISE since it can use a variety of goals to express the plan. The important square to move to may change depending on the opponent's move so it is not always possible to specify such a square in advance. For example, if white makes a move which traps and attacks black's queen then (after some desperado move by black) white would like to capture black's queen wherever it may be. PARADISE can express the plan of capturing the queen anywhere, while Pitrat's move statement cannot since the planner does not know which particular square the queen will be on.

Pitrat's modification statement is a goal which the system tries to accomplish. These modifications are too simple to express the purposes behind their suggestion. For example, the system may want to decoy the black queen to make a square that it protects safe for white. Pitrat would express this goal as removing an enemy from the black queen's location which does not express the purpose of the plan. It will work well for the winning combination but will also allow many wrong moves since the queen may be decoyed to a square from which she can still protect the square in question. PARADISE can avoid this since it has the ability to express its purposes. It can specify that the black queen must be decoyed to make the particular square safe, and only decoys which remove the black queen's protection will be considered. PARADISE would also specify that the move after the decoy should be safe so it will not be tried unless the preparations have accomplished their purpose. PARADISE attempts to express the purpose of each plan while Pitrat's plans express side effects that will happen if the purpose is accomplished (e.g., a square becoming vacant). Unfortunately, the same side effects may also happen when the purpose is not accomplished, causing the system to waste effort searching poor lines.

Plans in PARADISE have two more major advantages over the plans in Pitrat's system. First, PARADISE has conditionals which allow specification of different plans for different replies by the opponent. The advantages of this are obvious: the system can immediately try the correct plan instead of searching an inappropriate plan and backtracking to correct itself. Second, PARADISE's plan language is modifiable. Simply by writing new productions, new goals and concepts can be created for expressing plans and the system will automatically understand these new plans. This property is necessary for any system that wishes to extend its domain or incrementally increase its expertise. Significant additions to Pitrat's plan language would seem to require a major programming effort.

Despite the shortcomings of the plans in Pitrat's program, they are much more sophisticated than any plans previously used in computer chess programs. Pitrat's program performs well in its domain. It processes nodes much faster than PARADISE and therefore can handle larger trees. The plans do an adequate job given these constraints. It may be the case that programs will obtain better performance by using Pitrat's approach of larger trees and less sophisticated plans, but the use of knowledge is only starting to be investigated by programs such as PARADISE and it is too early to draw conclusions.

14. Summary

PARADISE exhibits expert performance on positions it has the knowledge to understand, showing that a knowledge-based approach can solve some problems that the best search based programs cannot solve because the solutions are too deep. The knowledge base is organized into KSs which provide concepts for PARADISE to use in its reasoning processes. These concepts are higher level than the ideas produced by simply matching patterns. A single pattern cannot recognize a whole plan of action in a complex chess position; both pattern-level ideas and these higher level concepts are necessary in the reasoning process.

The concepts provided by the KSs are used to create plans during static analysis. The concepts communicate enough information that they can be rejected after being posted. This keeps antithetical ideas from being combined to produce ridiculous plans. The same concepts are used to express plans for guiding the search, and the KSs execute the plans during the search. By communicating plans down the tree, PARADISE can understand new positions on the basis of its analysis of previous positions. By having particular goals in mind when looking at a new position, the system quickly focuses on the relevant part of the knowledge base. Plans in PARADISE express their purpose fairly well, without being too general or too specific. They contain enough additional information to enable the system to decide when a line has succeeded or failed. Because of the successful communication of knowledge through plans, PARADISE is able to solve a number of chess combinations with small trees and without a depth limit on its search.

PARADISE's knowledge base is amenable to modification. Production rules (and therefore KSs) can be quickly written and inserted in the knowledge base to improve system performance without adverse affects. Since KSs are used in both planning and static analysis, modifications can improve both without program changes. The range of expressible plans can easily be increased.

15. Issues

One of the major issues in creating concepts is their generality. How 'high-level' should a concept be? If the concepts are too general then too many details are lost, and the system does not have the necessary facts to do certain reasoning. If not general enough, the system leans toward the extreme of needing a production for every possible chess position. The concepts in PARADISE have been constructed to be as general as possible without sacrificing expert-level performance to the loss of detail. The factor limiting generality is the ability to communicate details in the attribute lists of concepts. The generality in PARADISE has made concepts in the data base so complex that productions must essentially match patterns in these concepts.

The tradeoff of generality and specificity in concepts is closely related to the tradeoff of search and knowledge. With much specific knowledge the use of knowledge becomes expensive, but fewer mistakes need to be corrected in the search. Such a knowledge-based approach can solve deep problems (e.g., PARADISE has no depth limit), but may not be able to solve many problems, even easy ones, because of holes in its knowledge. With more general knowledge, analysis is cheaper but more time must be spent correcting mistakes in the search. Such a search-based approach often leads to a program which does a full width search. Such programs almost always solve problems within their depth limit, but cannot solve deep problems and suffer from problems like the horizon effect (see Berliner [1]).

PARADISE uses a mix of search and knowledge which involves more knowledge and less search than previous programs which include chess middle games in their domain. The strength of the program depends on the completeness of the knowledge base. Programs like CAPS and Pitrat's do not search all legal moves and may also have holes in their knowledge. One might expect PARADISE to miss combinations more often than these two programs since its more specific knowledge suggests fewer moves, but this has not been the case in the comparison of CAPS and PARADISE on the test postions. PARADISE recognizes mating attacks that both CAPS and Pitrat's program miss. PARADISE's knowledge base appears complete enough to outperform CAPS on the first 100 positions in Reinfeld [9], even on positions within CAPS' depth limit. PARADISE-2's knowledge base is complete enough to outperform CHESS 4.4 on these problems.

The primary justification for the cpu time used by PARADISE and the biggest advantage of the approach used in PARADISE is the extendability of the knowledge base. This means the effect of holes in the knowledge can easily be reduced in PARADISE. The play of most programs cannot be noticeably improved with easily made program or knowledge modifications. Improved play would involve programming a more sophisticated analysis, or controlling the search better so that it could search deeper. (Play can, of course, be improved by switching to a significantly faster computer in most cases.) It would be hard to incrementally improve performance of a program by increasing its depth limit since each increment of one ply multiplies the effort required by the branching factor (usually about 5 or 6 for programs with full width searches that have a reasonable move ordering for alpha-beta). The possibility of incremental improvement through improved analysis is discussed below.

Programming a more sophisticated static analysis in any of the programs mentioned in this paper (except PARADISE) would probably not be easy for the programmer. Even if it were, these programs are committed (by tree size) to limit the amount of processing per node. CAPS spends more time per node than the others, but still processes five nodes per second of cpu time. CHESS 4.7 processes around 3600 nodes per second on a Cyber 176. If CHESS 4.7 spent even an additional ten milliseconds per node in analysis, the size of the tree it could search in a given amount of time would be greatly reduced. Easy incremental improvement of CHESS 4.7 (and similar search-based programs) by making analysis progressively more sophisticated does not seem possible. This means the theoretical limitations imposed by the depth limit cannot easily be overcome.

The time constraints on CAPS and Pitrat's program would permit slight increases in the time spent on analysis. Such slight increases cannot involve very specific knowledge, so any knowledge which recognizes a new tactic would suggest more moves to be searched in many positions. This increases the size of the search tree exponentially. These programs are already running efficiently and would either have to develop new search control mechanisms, or use significantly more cpu time to cope with this increase in branching factor.

For PARADISE's knowledge base to achieve the completeness needed to rival the best search-based programs (in middle game tactics), many productions may yet have to be added. There is no evidence that suggests large numbers of productions will harm PARADISE's performance (although problems may arise). New productions do not seem to significantly increase the branching factor when the added productions are at the right level of specificity (see Wilkins [13]). New productions do not significantly increase the analysis effort largely because plans guide PARADISE's search so well that a static analysis is done at less than nine percent of the nodes generated. Thus most new productions will rarely be executed. Even when executed, productions usually do not match and the system can often determine this with little effort. Even when matched, well-written productions take only a few milliseconds to a few hundred milliseconds to match. Compared to the 12 seconds of cpu time PARADISE spends on a typical analysis, this is not significant. The limiting factor in PARADISE's incremental improvement appears to be the ability to recognize more complex tactics at the correct level of specificity.

Weighing the above tradeoffs in order to judge which approach is best depends on the current technology. With developments such as faster

machines, parrallelism, or better hashing techniques, programs which rely on search can search deeper and perform better. The development of better pattern recognizers might increase the power of PARADISE's production language allowing PARADISE to perform better. There is little doubt that with current machines and our current understanding of how to use knowledge, good search-based programs solve a larger class of problems with less expense than do the best knowledge-based programs. However, techniques for using knowledge are only beginning to be developed and understood. This research shows ways to effectively use a large knowledge base which can be easily extended. As progress is made in representing and using knowledge, a knowledge-based program may eventually be able to approach (and even surpass) the performance of human chess masters.

ACKNOWLEDGEMENTS

The author is indebted to Hans Berliner for his continuing assistance and counsel during this research, and to the Stanford Artificial Intelligence Laboratory which provided the necessary environment and tools for doing this research.

REFERENCES

1. Berliner, H., Chess as problem solving: The development of a tactics analyzer, Unpublished doctoral thesis, Carnegie-Mellon University (1974).
2. Charness, H., Human chess skill, in: P. Frey (Ed.), *Chess Skill in Man and Machine* (Springer, Berlin, 1977), Chapter 2.
3. De Groot, A. D., *Thought and Choice in Chess* (The Hague: Mouton, 1965).
4. Davis, R., Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases, AIM-283, Computer Science Department, Stanford University, 1976.
5. Fahlman, S. E., A planning system for robot construction tasks, *Artificial Intelligence* **5** (1974). 1–49.
6. Gillogly, J., The technology chess program, *Artificial Intelligence* **3** (1972) 145–163.
7. Kotov, A., *Think Like a Grandmaster* (Chess Digest, Dallas, 1971).
8. Pitrat, J., A chess combination program which uses plans, *Artificial Intelligence* **8** (1977) 275–321.
9. Reinfeld, F., *Win At Chess* (Dover Books, 1958).
10. Sacerdoti, E. D., Planning in a hierarchy of abstraction spaces, *Artificial Intelligence* **5** (1974) 115–135.
11. Sacerdoti, E. D., The nonlinear nature of plans, Stanford Research Institute Technical Note 101 (January 1975).
12. Slate, D. and Atkin, L., CHESS 4.5—The Northwestern University chess program, in: P. Frey (Ed.), *Chess Skill in Man and Machine* (Springer, Berlin, 1977).
13. Wilkins, D. E., Using patterns and plans to solve problems and control search, AIM-329, Computer Science Department, Stanford University (1979).
14. Wilkins, D. E., Causality analysis in chess, *Proceedings of Third Conference Canadian Society for Computational Study of Intelligence, Victoria, May* 1980.