UMRAO: A Chess Endgame Tutor

Dinesh Gadwal Jim E. Greer Gordon I. McCalla
ARIES Laboratory
Department of Computational Science
University of Saskatchewan
Saskatoon, CANADA S7N 0W0

Abstract

Most research in computer chess has focussed on creating an excellent chess player, with relatively little concern given to modelling how humans play chess. The research reported in this paper is aimed at investigating knowledge-based chess in the context of building a prototype chess tutor, UMRAO, which helps students learn how to play bishop-pawn endgames. In tutoring it is essential to take a knowledge-based approach, since students must learn how to manipulate strategic concepts, not how to carry out minimax search. UMRAO uses an extension of Michie's advice language to represent expert and novice chess plans. For any given endgame the system is able to compile the plans into a strategy graph, which elaborates strategies (both well-formed and ill-formed) that students might use as they solve the endgame problem. Strategy graphs can be compiled "off-line" so that they can be used in real time tutoring. We show that the normally rigid "model tracing" tutoring paradigm can be used in a flexible way in this domain.

1 Introduction

Chess playing has been the subject of intense investigation by human practitioners, cognitive psychologists, and by those in the field of artificial intelligence (AI). Chess has the potential to be a "Drosophilia" (or fruitfly) for AI and cognitive science [Simon and Chase, 1973], because of its nature as an intellectual activity. In trying to create a superior computer chess program most AI researchers have concentrated on improving search techniques. There have been some knowledge-based chess projects which have explored more human ways of choosing moves, but they are unable to compete against the search-based approaches. Current knowledge-based programs use the concepts of chunking [Berliner and Campbell, 1984], advice [Michie, 1977], and plans [Pitrat, 1977; Wilkins, 1980] to represent chess knowledge. Although search-based programs are successful from a performance viewpoint, they make little contribution to the most interesting objectives set in choosing chess as an AI problem; that is, modelling intelligent behaviour within a well defined domain. The problem lies with the objective of creating a chess player: the goal has been achieved without resolving the fundamental questions.

This paper attempts to define the problem of developing an intelligent tutoring system for a sub-domain of chess. This change of goals requires a shift of emphasis which illustrates and illuminates many problems that arise in AI. Chess tutoring requires the resolution of cognitive and epistemological issues inherent in knowledge-based chess. As a first step towards building a chess tutor, a prototype system, UMRAO, has been developed for the limited domain of bishop-pawn chess endgames. A set of 22 problems involving two white pawns, a black bishop, and two kings have been used [Averbach, 1980] for system development. Figure 1 illustrates one such endgame.

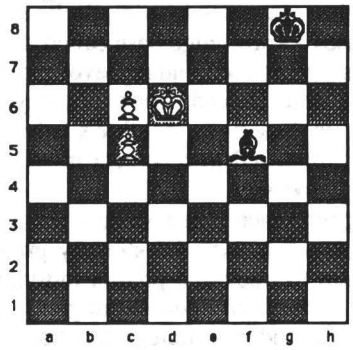

Figure 1 White to Play and Win [Averbach, 1980]

Although, chess endgames are simpler than other parts of the game, they are still complex enough for interesting tutoring. Endgames are amenable to a knowledge-based approach [Michie, 1977]. Chess is also a suitable domain for intelligent tutoring system (ITS) research, being reasonably complex, yet at the same time having well-formed solutions consisting of various interacting strategies. Chess is not as simple as ITS gaming domains like WEST [Burton and Brown, 1982], nor as complex as ITS programming domains such as SCENT [McCalla et al., 1988]. An intelligent chess tutoring system has not been developed before and may prove to be a better learning tool than chess playing programs and standard chess books.

2 System Design and Methodology

The system has been developed in two phases. In the first phase, efforts were made to understand novice chess skills. Novice chess players (players sufficiently versed in chess to begin making strategic decisions) were informally "tutored"

We would like to thank the Natural Sciences and Engineering Research Council of Canada for their financial support of this research.

in order to collect think aloud problem solving protocols. Each was asked to solve problems from a set of twenty-two bishop pawn endgames and meanwhile their problem solving behaviour and human expert tutoring behaviour were recorded. The outcome of these protocols was the identification of the important design requirements for UMRAO. In the second phase, the system was designed and implemented according to the model of chess problem solving behaviour we discovered. The following design requirements for UMRAO were identified:

- The system must be able to model correct and incorrect solution strategies.
- Instructions should be given in context.
- Feedback should be conceptual, using higher level concepts such as plans and goals instead of simple feedback specifying move correctness or giving the next correct move.
- Students should be given an opportunity to explore incorrect strategies.

3 The Architecture

UMRAO consists of two parts: the EXPERT and the TUTOR. The EXPERT is run only once for each new bishop-pawn endgame problem. For each such problem the EXPERT compiles a strategy graph representing plausible student strategies and sub-strategies as well as expert-level counter strategies. In order to compile its strategy graph, the EXPERT uses a knowledge base called the plan library, which contains expert and student level plans in order to model correct and flawed lines of play. The plan representation is based on Michie's advice definition [Michie, 1977] modified to represent misconceptions and faulty reasoning processes.

The TUTOR runs each time a student wants to work on a particular problem. It uses the strategy graph associated with the problem to track and predict individual students' moves in the context of their strategies. Because the TUTOR tracks the student's strategy (or strategies), it can, at any point, comment on strategic shortcomings. It is also able to generate explanations contrasting expert and student strategies. Note that the process of generating an explanation is efficient, since the strategy graph has been generated off-line, before the tutoring session. The TUTOR tracks through the graph; it does not build it. The basic approach is model tracing as in Anderson et al. [1990], but we have been experimenting with two instructional styles: the usual immediate feedback, as well as delayed feedback where there is scope for user initiative.

3.1 The EXPERT Module

The EXPERT can be explained in terms of its *plan representation*, its *strategy graph* and its *planning system*. The representation of plans is the back bone of UMRAO because of its importance in defining expert, student, and ill-formed strategies in one framework; in providing the basis for generating feedback using higher level concepts such as goals, constraints, and patterns; and in guiding interaction by suggesting situations appropriate for tutor intervention. In UMRAO, a plan is an object with slots representing various aspects of the plan: *side to play* (black or white), *type of plan* (expert or student), *applicability* predicate, *feasibility* predicate, *better-goals*, *holding-goals*, *move-constraints* for both sides, and *decidability* of the plan.

The slots for *applicability*, *feasibility*, and *decidability* are additional features not present in Michie's advice language. An *applicability* predicate consists of board features, such as the pawn formation (blocked or passed pawns), which favour selection of the plan. *Feasibility* predicates indicate the potential success of a plan. The criterion of success or the purpose of the plan is described by its *better-goals*. The criterion of failure of the plan is defined by its *holding-goals*. For example, if a side has a passed pawn (*applicability*) and is not controlled by the bishop (*feasibility*) then the plan can be to 'queen the pawn' (*better-goal*), and at the same time the pawn should be safe (*holding-goal*), and the success of the plan decides the game (*decidability*). The *move-constraints* are represented as Mc_x and Mc_y, where Mc_x defines the constraints on the moves for the side to play and Mc_y defines the constraints for the opponent's moves in order to satisfy the goals (*better-goals* and *holding-goals*) of the plan.

Figure 2 shows examples of a student level and an expert level plan applicable in the board position shown in Figure 1.

A student level plan

```
((Identification W5)          (ToPlay (white student))
(Applicability (and (double-pawn P1 P2)
                    (not (feasible W1))))
(Feasibility   (can-support WK (bishop-square P1)))
(Better-Goal   (and (exchange P1 B) (queen P2)))
(Holding-Goal  (and (safe WK) (safe-from P1 B) (safe P2)))
(Mcx           ((destination P1 (queening-square P1))
               (destination WK (near (bishop-square P1))
               (destination P2 (promotion-square P2))))
(Mcy           nil)
(Decidability  t))
```

A corresponding expert plan

```
((Identification W6)          (ToPlay (white expert))
(Applicability (and (black-king-threatens P2)
               (double-pawn P1 P2) (not (feasible W1))))
(Feasibility   (can-support WK (bishop-square P1))
               (can-prevent WK BK P2)))
(Better-Goal   (and (exchange P1 B) (queen P2)))
(Holding-Goal  (and (safe WK) (safe-from P1 B) (safe P2))
(Mcx           ((destination P1 (queening-square P1))
               (destination WK (near (bishop-square P1)))
               (destination P2 (promotion-square P2))))
(Mcy           ((move-toward BK P2)))
(Decidability  t))
```

Figure 2 Example of Expert and Student Level Plans

In commonsense terms the student plan says: "If you have double pawns and the first pawn is prevented by the bishop from queening: Exchange the first pawn with the bishop and queen the second pawn." The expert level plan says: "If you have double pawns and the first pawn is prevented by the bishop from queening and the opponent's king can reach the second pawn: Exchange the first pawn with the bishop and

queen the other pawn, but in the mean time keep the black king away."

The plan representation can be better explained by considering the expert plan W6 in Figure 2. The first *applicability* feature (black-king-threatens P2] is a geometric relationship between the second pawn and the black king. The second feature is (double-pawn P1 P2) which matches when there are two pawns in the same column. The third feature is (not (feasible W1)) which matches when a particular plan W1 is not feasible. This feature specifies a kind of plan ordering. What it means is that if plan W1 is not feasible only then should plan W6 be considered. In this way procedural knowledge has been coded in a declarative fashion. This ability to represent a variety of procedural knowledge in a declarative form adds to the generative capability of the system.

The *feasibility* slot of plan W6 contains two features. The first feature (can-support WK (bishop-square P1)) contains the primitive feature (bishop-square P1), which represents the square common to P1 and the Bishop. This ability to make composite features also adds to the generative capability of the representation. The second feature used in the given *feasibility* slot is (can-prevent WK BK P2) which matches if WK (the white king) is able to prevent BK (the black king) from reaching near P2 (the second pawn).

The move-constraints *Mcx* and *Mcy* generally specify the constraints on the movement of a various pieces of plan's side and the opponent's side, respectively, in order to execute a particular plan. The movement of the piece is specified in terms of a square (or a group of squares) having a particular feature. For example, the three move-constraints in *Mcx* direct the movement of: pawn P1 towards its queening square, the white king towards one of the squares near the square on the pawn's path controlled by the bishop, and pawn P2 towards its promotion-square. The single move-constraint in *Mcy* specifies to limit the ability of the opponent's king to move towards pawn P2.

The differences between the expert and student level plan in Figure 2 begin with the *applicability* conditions of the plans. The student has not considered/perceived the (black-king-threatens P2) relation and as a result does not check for the *feasibility* of the black king capturing the second pawn, which produces a less adequate set of *move-constraints* (Mc_x and Mc_y). This in turn is responsible for the deviation from the correct solution strategy.

A variety of student level plans can be generated easily by perturbing expert level plans. For example, novices frequently do not check for the *feasibility* of an otherwise correct plan. This can be modelled by creating a new plan, *Wx'* , by weakening the *feasibility* slot of plan *Wx*. Of course, not all such syntactic deviations are meaningful and they must be supported by empirical observation.

3.1.1 The Strategy Graph

The strategy graph is a knowledge source for the TUTOR module made of nodes and links. The nodes of a strategy graph are called *cnodes*, *strategies*, and *move-plan* objects. The analysis contained in a strategy graph is made accessible to the TUTOR by placing the strategy graph's *root cnode* in the applicable problem object. The *root cnode* represents the initial state of the problem (initial board position), all the plausible strategies for that position, and the corresponding moves. At the *root cnode* it is always the student's turn to make a move.

Figure 3 shows a portion of the strategy graph for the chess endgame problem given in Figure 1. Each *cnode*, labelled Ci and shown by a rectangular box in Figure 3, is connected to its descendant *cnodes* by a move justified by a set of plans. The strategy graph cnodes are classified as either *student* or *expert cnodes* depending upon who has to make a move at that cnode. Each *student cnode* is linked to a set of *expert cnodes*, or it is a terminal *cnode*. Similarly, each *expert cnode* has a single *student cnode* descendent, or it is a terminal *cnode*. Each *cnode* contains links to applicable strategies. For example, in Figure 3, C0 is connected to C2 by a move m1 justified by a set of plans W1 W5 W6. Although all the *cnodes* shown in Figure 3 have only a single parent, in complete strategy graphs for problems in UMRAO, many *cnodes* have multiple parents. It is important to notice that at all the expert *cnodes* (levels where it is the system's turn to make a move) there is only one descendant, the best move. This set of *cnode* objects, connected with *Descendant* and *Parent* links, forms the skeleton of the strategy graph.

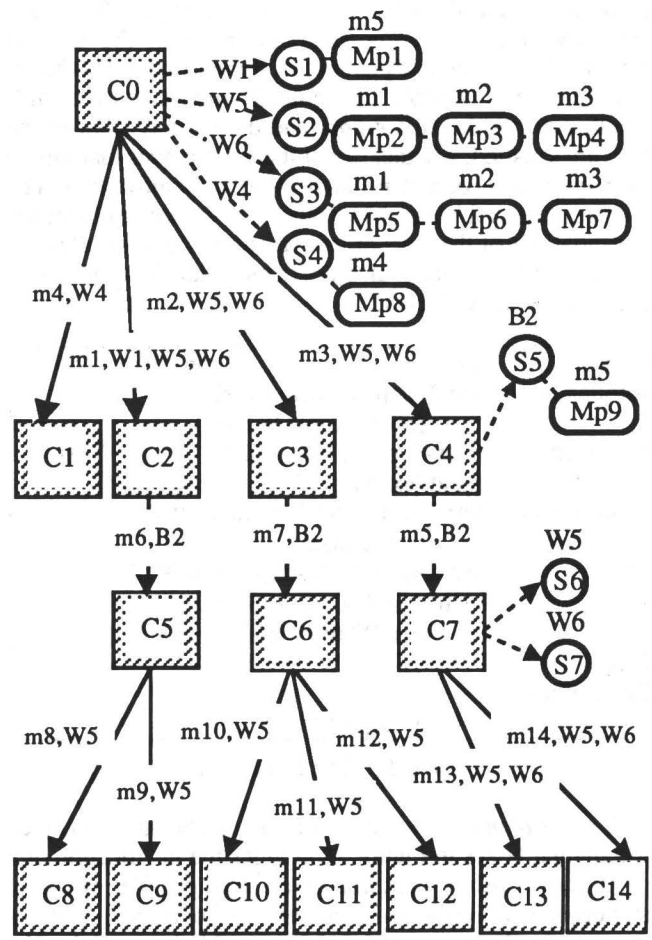

Figure 3 The Strategy Graph

A *cnode* represents all the information available to the TUTOR during the process of monitoring the student's solution and conducting a tutoring session. A *cnode* contains the board configuration, information as to which side has the move, the parents and descendants of the *cnode* (lists of (move, *cnode*) pairs), pointers to all applicable strategies and the possible moves that correspond to these strategies, the descendant *cnode* corresponding to the best response, and the best result obtained by the *cnode*. Figure 4 shows the instantiated *cnode* C0.

Identification:	C0
Configuration:	(initial board position shown in Figure 1)
SideToMove:	White - Student
Parent:	nil
Descendants:	((m4 C1) (m1 C2) (m2 C3) (m3 C4))
Strategies:	(S1 S2 S3 S4)
Moves:	(m1 m2 m3 m4)
BestResponse:	((m3 S3))
Result:	(White nil)

Figure 4 The Cnode C0

Each *cnode* object has associated with it a set of *strategy* objects Si, shown by circles in Figure 3. These *strategy* objects are instantiations of applicable plans. For example, *cnode* C0 has a set of strategies S1, S2, S3, and S4 which are instantiations of plans W1, W5, W6, and W4 respectively (strategy S2 is detailed in Figure 5). Each *strategy* object contains information about the various aspects of an applicable plan object, for example, whether or not the plan is feasible (designated with values t or nil), whether or not the *better-goal* and the *holding-goal* of the plan are satisfied, whether or not there are moves that satisfy the requirements of a plan, and how different moves are rated with respect to a plan. Most slots of a *strategy* object are instantiations of the slots of its corresponding plan. Besides these, a *strategy* object also contains a slot that points to a list of *move-plan* objects.

Identification:	S2 From Plan: W5
Applicability:	((and (double-pawn P1 P2) (not (feasible W1))) t)
Feasibility:	((can-support WK (bishop-square P1)) t)
Better-Goal:	((and (exchange P1 B) (queen P2)) nil)
Holding-Goal:	((and (safe WK) (safe-from P1 B) (safe P2)) t)
Mc_x:	((destination P1 (bishop-square P1)) (destination WK (near (bishop-square P1))) (destination P2 (promotion-square P2)))
Mc_y:	nil
Decidability:	t
MovePlanObj:	(Mp2 Mp3 Mp4)

Figure 5 The Strategy S2

A *move-plan* object, labelled Mpi and shown by rounded rectangular boxes in Figure 3, provides the justification of a given move with respect to a given strategy. For example, strategy S2 (which is an instantiation of plan W5) contains three *move-plan* objects Mp2, Mp3 and Mp4, containing the justification of moves m1, m2, and m3 respectively with respect to strategy S2. Each *move-plan* object contains the analysis of a plan after a given move has been made. For example, after making a move (let's say m1), this analysis will determine if the plan (W6) is feasible for the resulting position; in addition, it reveals the evaluation of the move m1, with respect to the move-constraints of the plan W6. Each move-plan has an associated move. The slots of a *move-plan* object are detailed in Figure 6. Values associated with each move constraint reflects the progress the move makes towards meeting that constraint. An overall evaluation of the move is derived from the values of the move constraints. The next subsection explains how these different objects that build a strategy graph are generated.

3.1.2 The Planning System

The planning system used for generating the student and expert problem solving behaviour is based on depth-first search. Alhough the methodology has an element of search, it is quite different from the methods used in the usual search-based chess programs. Search in those programs is over the space of possible moves while in UMRAO search is over the space of plausible strategies, i.e., UMRAO is based on planning rather than look-ahead.

The way planning is implemented in UMRAO is quite different from the planning in usual knowledge-based chess programs. Typical knowledge-based chess programs have to analyze only the best or most promising plans at any time. They do not have to look for other plans until the current plan is abandoned for some reason. In contrast, UMRAO has to analyze all plausible plans at every student *cnode*. This is because the planning behaviour of students differs from that of an expert. In other words, all the applicable plans have to be analyzed for their consequences. Even for the *expert cnodes*, static analysis is carried out at every *cnode*. Because of this additional requirement of analyzing less than optimum plans, a different design for the planning system had to be devised. One important characteristic is the ability to simultaneously analyze different plans, which makes it suitable for parallel implementation.

Identification:	Mp4
Move:	m3 Plan: W5
Better-Goal:	((and (exchange P1 B) (queen P2)) nil)
Holding-Goal:	((and (safe WK) (safefrom P1 B) (safe P2)) t)
Mc_x:	(((destination P1 (bishop-square P1)) 0) ((destination WK (near(bishop-square P1))) 1) ((destination P2 (promotion-square P2)) 0))
Mc_y:	0
Evaluation:	1

Figure 6 The Move Plan Object Mp4

The methodology consists of two algorithmic modules: ExSearch and StSearch. The ExSearch module is applied when the EXPERT has to generate and analyze the system's response in a given board position, while StSearch is applied when the EXPERT devises possible student responses. ExSearch is very similar to standard knowledge-based chess programs. Once ExSearch finds a suitable response, it does not have to look at the remaining possible responses, since it only has to generate the best counter

response for a given student's response. StSearch must analyze all the plausible student responses exhaustively, as the tutor should be prepared for a range of student responses to a board position. Further, ExSearch only considers plans that are of Expert level while StSearch has to analyze all kinds of plans. ExSearch does not consider plans that are not feasible while StSearch considers all applicable plans irrespective of their feasibility. ExSearch analyzes all the moves that correspond to at least one feasible plan until it finds one that results in a favorable position for the system and then stores the analysis of the best move, while StSearch analyzes all plausible moves and stores the analysis of all of them. Detailed algorithms for both StSearch and ExSearch can be found in [Gadwal, 1990].

3.2 The TUTOR Module

While the EXPERT generates all the chess expertise required to help a student with a particular chess endgame problem, the TUTOR actually carries out the tutoring. The student can choose to play any of the endgame problems from the problem library. The goal of the TUTOR module is to challenge students to solve new problems while monitoring and commenting upon their actions. The system can recognize optimal, less than optimal, or clearly irrelevant moves. The student continues problem solving while the TUTOR offers help, hints, explanations, and tutoring advice when needed or requested. The main pedagogical goal underlying the design of the TUTOR module is to be a partner and co-solver of problems with the student, who is encouraged to experiment with various strategies.

A variety of tutoring styles can be implemented with the tools provided in UMRAO. Four tutoring styles have been implemented and tested. In *immediate feedback with strict tutor control*, UMRAO immediately explains the student's strategy and describes a more suitable strategy as soon as the student makes a suboptimal move. This is similar to the "model tracing" used in many of Anderson's tutors [Anderson et al., 1990] In *immediate feedback with student initiative*, UMRAO alerts the student when a suboptimal move is made, but offers options for the student to explore a faulty line of play, request a hint, or an explanation of the current suboptimal strategy or the best strategy. *Optional feedback with strict tutor control* does not allow the student to deviate from the path of optimal play, but satisfies students who want the tutor to provide explanations only when requested. *Optional feedback with student initiative* provides the student with a full set of options including *hint*, *best move*, *explain strategy*, *tell*, *try another* (take back) and *play on*. These options are selected by the student whenever feedback is desired. Once play reaches a terminal position, that is, an obvious win or loss for one of the players, an obvious blunder by the student, or a move that deviates from any known strategy, the system always takes initiative to force a choice.

To give a flavour of the tutoring interaction that has been implemented, consider a sample tutoring session for the bishop-pawn endgame originally presented in Figure 1. Figure 7 shows some of the screens taken from an actual tutoring session with UMRAO using immediate feedback with student initiative as the tutoring option. The TUTOR presents a selected endgame problem to the student and asks for a white move. The student makes a move by dragging a piece on the board from one square to another. The system updates the board and tells the student that the chosen move is not the best move and provides a set of menu options (shown in Figure 7a). Figures 7b and 7c show some of the types of feedback provided by the system in response to a request for a hint and for a full explanation of the winning strategy.

Suppose that the student ignores the system's advice and decides to pursue a line of suboptimal play. The system reacts to the student's move, and after some time, a position is reached where it becomes obvious why the first move was not correct (shown in Figure 7d). Play then returns to an earlier board position where there is a way for the student to correctly solve the problem. In the complete session, as the student tries out various moves, the TUTOR tries to recognize the plan behind these moves and offers a variety of feedback. This allows the student to explore both correct and incorrect strategies.

4 Conclusion

There are a number of specific technical contributions of UMRAO. UMRAO extends traditional notions of knowledge-based chess. It necessitates the incorporation of poor strategies as well as good ones into the knowledge base. UMRAO also has an interesting hybrid planning technique that is like traditional knowledge-based approaches in its search for optimal plans when it is the expert's turn to move, but is original in its need to consider all plausible plans when it is the student's turn to move. Moreover, UMRAO delineates an elegant separation between plans and strategies. Plans are problem-independent; strategies are problem-specific. This separation is used by UMRAO so that it can compile from the corpus of possible plans a particular strategy graph that is tailored to each endgame problem The inefficient compilation of strategy graphs can be done "off-line" so that they can later be efficiently used in real time student-tutor interaction.

UMRAO also makes specific contributions to intelligent tutoring. UMRAO shows how the model tracing tutoring methodology can be made flexible and responsive to the student, instead of rigid and dictatorial as it is often perceived to be. UMRAO also provides a laboratory for experiments in various kinds of tutoring strategies: the system's deep understanding of chess strategies allows tutor-controlled or student-controlled pedagogy; and allows a choice between immediate and delayed feedback to the student.

Finally, there are general contributions of this research that transcend particular subdisciplines of artificial intelligence. The tutoring domain re-establishes chess as a natural exploratory environment for ideas other than search, and provides the area of intelligent tutoring systems with a perfect domain for exploring deep ideas in diagnosis and pedagogy without the complexities of other domains. With further development UMRAO may also prove to be a practical contribution to the teaching of chess, more responsive and adaptable than a book, an expert that not only can present chess problems to students, but also solve them and comment strategically on them.

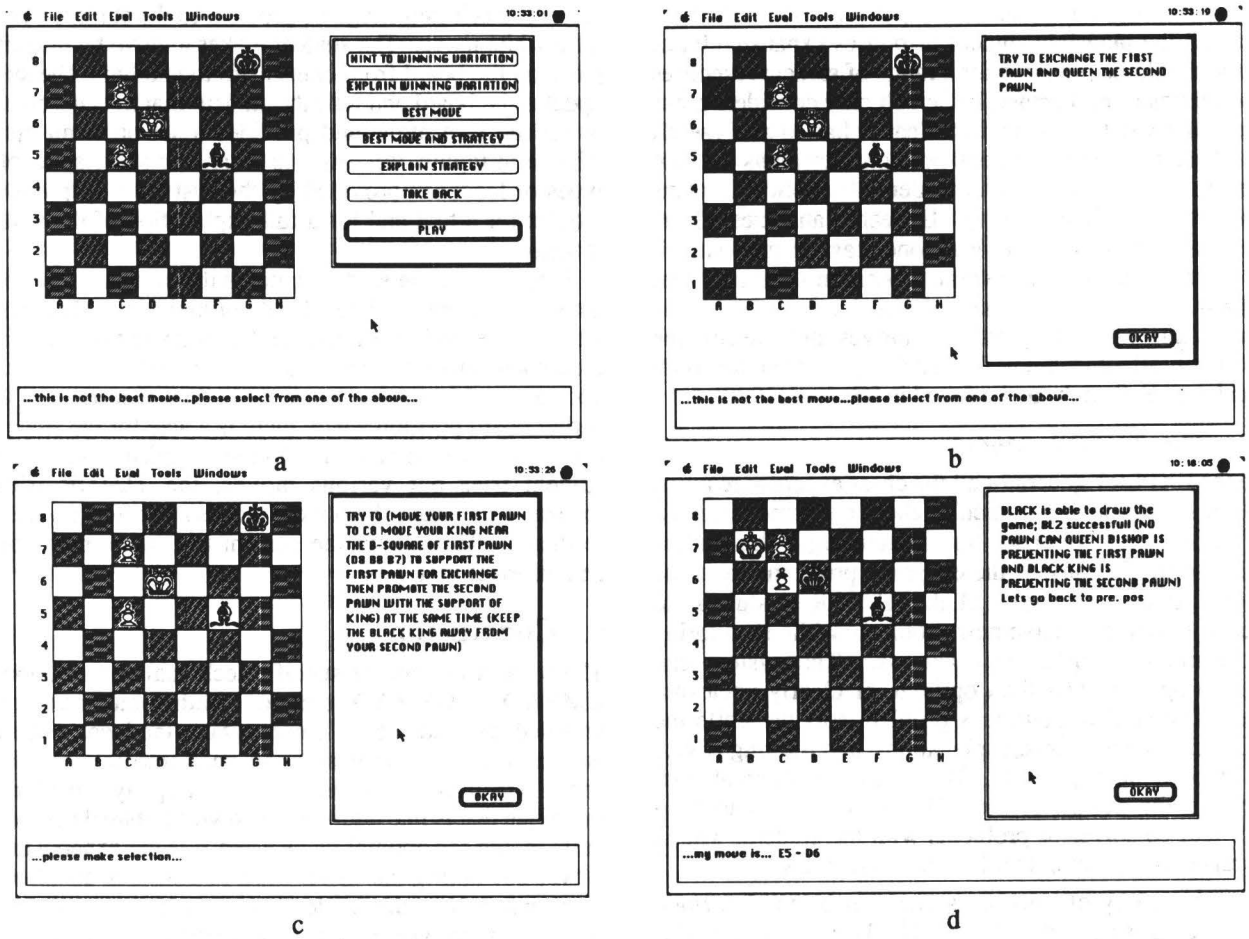

Figure 7 Sample Screens from UMRAO

UMRAO is not yet a complete chess tutor, but it represents a good starting point for future research in ITS, cognitive science, and knowledge-based chess. Such a chess tutor can act as a testbed for various theories about expert and novice chess skills, for testing techniques in student modeling and tutoring strategies, and for exploring the representation and reasoning schemes of knowledge-based chess. The strengths of the implementation are its modularity and simplicity, qualifying it to be a good experimental system for exploring various ITS issues.

References

[Anderson et al., 1990] J.R. Anderson, F. Boyle, A. Corbett & M. Lewis. Cognitive Modeling and Intelligent Tutoring, *Artificial Intelligence*, 42, 7-49, 1990.

[Averbach, 1980] Y. Averbach. *Comprehensive Chess Endings (Vol 1)*, Academic Press, New York, 1980.

[Berliner and Campbell, 1984] H. Berliner and M. Campbell. Using Chunking to Solve Chess Pawn Endgames, *Artificial Intelligence*, vol. 23, 97-120, 1984.

[Burton and Brown, 1982] R. Burton and J.S. Brown. An Investigation of Computer Coaching for Informal Learning Activities, in Sleeman and Brown, (Eds.), *Intelligent Tutoring Systems*, Academic Press, New York, 1982.

[Gadwal, 1990] D. Gadwal. *UMRAO: A Chess Endgame Tutor*, M.Sc. Thesis, Department. of Computational Science, University of Saskatchewan, 1990.

[McCalla and Greer, 1988] G.I. McCalla and J.E. Greer. Intelligent Advising in Problem Solving Domains: the SCENT-3 Architecture, *International Conf. on Intelligent Tutoring Systems*, Montreal, 124-131, 1988.

[Michie, 1977] D. Michie. A Theory of Advice, In Elcock and Michie (Eds.), *Machine Intelligence 8*, Edinburgh University Press, Edinburgh, 30-59, 1977.

[Pitrat, 1977] J. Pitrat. A Chess Combination Program Which Uses Plans, *Artificial Intelligence*, 8, 275-321, 1977.

[Simon and Chase, 1973] H. Simon and W. Chase. Skill in Chess, *American Scientist*, 61(4), 394-403, 1973.

[Wilkins, 1980] D. E. Wilkins. Using Patterns and Plans in Chess, *Artificial Intelligence*, 14, 165-203, 1980.