THE UTILITY OF EXPERT KNOWLEDGE

Jonathan Schaeffer
TA. Marsland

Computing Science Department,
University of Alberta,
Edmonton,
Canada T6G 2H1

ABSTRACT

How useful is the knowledge we add to an expert system? What is important knowledge? Can too much knowledge be bad? These questions are examined by presenting the preliminary results of experiments that paired programs with varying amounts of chess knowledge against each other. The experiments illustrate problems of interacting knowledge and give some insight into methodologies for "teaching" expert systems.

1. Introduction

With the increasing awareness of the potential for expert systems, knowledge engineering has become a recognized discipline. The addition of knowledge to an expert system raises some important questions: What should one add? How much? Can too much knowledge be a bad thing? The goal of answering these questions is to find a methodology for adding knowledge to an expert system, one that maximizes performance while minimizing redundancy and inefficiency.

Consider the analogy between a student and an expert system. Both go through a period of learning in which the objective is to raise abilities to a desired level of competence. However, the student attends schools in which the curriculum is organized so that new knowledge builds upon the old. It would be absurd, for example, to teach quantum mechanics as part of a grade 1 course. In contrast, expert systems are "taught" in an *ad hoc* manner. There is no established method for teaching an expert system, nor guidelines for organizing a curriculum.

In this paper, a series of experiments with chess knowledge is reported illustrating some of the difficulties with adding knowledge to an expert system. Many of the problems are analogous to those a student encounters when enrolled in a poorly designed education program. The solutions are often similar to the way they are solved by educators. The results of the experiments give some insight into the difficulties of "teaching" expert systems.

t A competitor in the most recent World Computer Chess Championship [1].

2. Experiment Design

A chess program has two distinct parts; the framework for making and analyzing moves, and the knowledge that allows the program to play well. The former includes legal move generation and tree searching, and is well understood. The latter, however, is vague and informal; it is the product of centuries of experiences that have been condensed into rules and exceptions, few of which can be formalized.

The expertise in our chess program *Phoenix* t was partitioned into the following 8 routines: Tactics (T), Space and Mobility (SM), Pawn Weaknesses (PW), King Safety (KS), Center Control (CC), Pawn Structure (PS), Incremental Scores (IS), and Planner (PL). Details of the contents of the routines can be found elsewhere [2] and are not essential to the points raised in this paper. To determine the utility of this knowledge, a pair of experiments have been performed; one to show how the knowledge should be acquired and one to see the consequences of its removal. The experiments illustrate the (un)importance of the knowledge routines as they interact with each other.

Each experiment followed an established technique [3-5] and consisted of a series of matches between versions of *Phoenix* with differing amounts of knowledge. A match consisted of 20 games, with each opponent playing the white and black side of 10 starting positions. The accumulative knowledge experiment starts initially with the basic tactics program, *T,* and uses it to play a series of matches against *T* supplemented with a different knowledge routine for each match. This allowed us to measure the effectiveness of each expert component relative to a program with no such knowledge. This process was repeated by gradually expanding the basic program *T* with more and more knowledge and using it to identify the next best piece to acquire. The removal experiment starts with *Phoenix* and, using the same technique, gradually eliminates knowledge. This allows us to measure the importance of the routines relative to a program with complete knowledge.

To limit the scope of the problem, we have restricted our attention to the acquisition of middlegame knowledge. It was therefore necessary to remove any influence that other phases of the game might have. Opening specific information was reduced by chosing diverse starting positions for the matches that were each ten moves into the game. Tree searching techniques were factored out of the experiment by having all programs use the same parameters and search to a depth of 5 ply. A game was considered over when it either ended in checkmate or draw, or when *Phoenix* determined that an endgame had been reached. In the latter case, the final position was adjudicated.

A difficult problem was posed by cases where there was no clear-cut winner, but one side had accumulated positional advantages that in the long-term may prove decisive. Since the opponents have slightly different models of what is important, it was possible for both sides to think that they had the advantage! The notion of a superior or inferior position was introduced to ensure that the advantage of long term factors could be considered even though the material balance was equal. To ensure impartiality, the adjudications were performed by the chess program Cray Blitzt. Adjudication resulted in a position being assessed a value in the range 0 to 1 with a win (worth 1 point) defined as the side to move being up a full pawn. If the position was not won, a value was assigned reflecting how (un) favourable the position was, with a value of 0.5 for a balanced position.

3. Results

The results of the accumulative knowledge experiment are summarized in Table 1. Each row gives the result of the matches between a base variant of *Phoenix (T* with 0 or more pieces of knowledge added) and that program supplemented by the piece of knowledge specified in the column heading. For example, the entry in row 3 and column 6 says that the base program *T + SM + CC* lost by a score of 9.35 to 10.65 to *T + SM + CC + KS.* Note, however, that the acquisition of knowledge cannot be done in an arbitrary manner. Consider row 3 column 7, where the base program won by a score of 11.15 to 8.85 over itself supplemented by PS. This, and similar apparent anomalies, illustrate that the haphazard addition of knowledge may not be effective until necessary basic knowledge is in place.

These results can be put into a more familiar form by expressing them as chess ratings, which provide a convenient means of equating the program's performance with human abilities. The average club player has a rating of about 1400. The details of the rating formula are not important and are discussed elsewhere [2]. Pegging the basic *T* program with a rating of 1110, experimentally determined and consistent with others [3], yields Table 2. Note that the version of Phoenix used has a tournament rating of 1840, close to the predicted 1786.

The tables support the well-known result [6] that the most important heuristic is Space and Mobility, since it gives *Phoenix* almost half its rating points. Space and Mobility is the simplest routine to implement and requires no real expert knowledge. In some sense, *SM* can be viewed as the first lesson in the education of a chess program.

After *SM,* it appears that the law of diminishing returns takes over. Additional knowledge provides fewer rating points for increased effort. It is interesting to note that the three smallest gains *(IS, PL,* and *PS)* were for the three largest routines containing the most heuristics. The inclusion of *IS* appears to have a *negative* effect on performance, although the difference between 9.65 and 10 is not significant (as verified by other experiments). This is an example of knowledge used as a *building block) IS* by itself may not be significant, but its presence provides the environment necessesary for effective use of subsequent additions. The rating gains obtained by adding knowledge appear to decrease steadily, except for King Safety (see Table 2). This anomaly may be explained by the observation that *KS* is not an important factor in most positions, and in many games has little bearing on the play.

Table 3 presents the results of the diminishing knowledge experiment. Whereas *SM* is the first piece of knowledge that one would give to a program, *PW* is the most valuable to retain. When *Phoenix* is supplemented by all the knowledge routines, *PW* plays a much more important role than it does when working in an environment with little knowledge. This illustrates that some knowledge needs the right environment with which to interact to achieve best results. An analogy might be teaching new material to students who do not have the proper pre-requisites.

The results of matches involving *PS* in both Tables 1 and 3 are interesting in that by removing *PS* the program often plays better! Most of the knowledge in *PS* is sophisticated, in the sense that it builds on many elementary concepts that would be taught early in any chess education. One possible explanation of the results then is that *Phoenix* does not know enough to use *PS* properly. Another possibility is that *PS* has not been implemented correctly; either there is a bug in the routine, or the knowledge has not been properly

t A portable version of the current World Computer Chess Champion (I], but running on a VAX 11/780.

represented. Regardless, it appears that *PS* may not be of significant benefit to *Phoenix* as it currently stands. The argument for inclusion of a piece of knowledge should take into account the expected benefit versus the cost in terms of space, execution time, and implementation time.

Finally, a few words of caution. These results must be taken in the proper perspective. They could be implementation dependent; other interpretations of chess knowledge may differ. In addition, since the experiments were done with only one program, the ratings reported should be interpreted as measuring *relative* rather than *absolute* importance. Each match consisted of 20 games and took an average of 55 hours of computing time on a VAX 11/780. Each experiment consisted of 28 matches for a total of 1120 games, taking over 6 months to complete. Despite the large outlay of computing resources some statistical variability is still to be expected.

4. Conclusions

Our experiments illustrate some of the benefits of a complete retrospective study of the knowledge in an expert system. In particular, they show that knowledge cannot be added in an arbitrary manner, since more sophisticated concepts require that certain fundamental ideas be in place before they can become effective. If the knowledge components are independent, they may be added in any order, but this is not normally the case. Rather, there are interactions which are used to resolve contradictions, in order to provide a "best guess decision" when the situation is not clear-cut. These interactions are seen clearly in the knowledge removal experiment. For example, they show the benefits of planning in chess (PL), something that was not obvious in the accumulation experiment. They also illustrate the problems with knowledge that is used infrequently (KS), or perhaps is not well understood (PS). All too often such knowledge may be handling special cases only, and may even be detrimental when applied inappropriately. Probably this is an indication that too much diverse knowledge is embodied into a single routine, leaving open the possibility that the best implementation of the ideas contained therein has not been provided.

Our experiments have opened up some other avenues of research. In particular, to what extent does knowledge compensate for depth of search? A well-known result is that an extra ply of search in a chess program is worth about 250 rating points [5], To what extent are the two interchangeable; can additional knowledge in *Phoenix* be used to compensate for a shallower search tree? Other experiments being formulated are designed to measure the granularity or (in)dependence of knowledge. None of the knowledge routines is completely independent of the others. By breaking the routines into finer granules, the interactions can be identified more clearly, and redundant or contradictory relations can be eliminated.

Base Program	Knowledge Removed						
	- PW	- SM	- PL	- CC	- IS	- KS	- PS
Phoenix	5.95	6.05	9.35	10.30	11.50	7.75	10.70
Phoenix	- PW	5.80	10.40	8.20	10.25	9.25	9.30
Phoenix	- PW	- SM	7.30	7.35	10.75	9.10	12.75
Phoenix	- PW	- SM	- PL	6.25	10.25	11.15	7.75
Phoenix	- PW	- SM	- PL	- CC	5.90	8.85	10.10
Phoenix	- PW	- SM	- PL	- CC	- IS	8.65	11.00
Phoenix	- PW	- SM	- PL	- CC	- IS	- KS	5.75
= Tactics							

Table 3: Diminishing Knowledge Results

Base Program	Knowledge Added						
	+ SM	+ CC	+ PW	+ IS	+ KS	+ PS	+ PL
Tactics	17.60	15.25	11.60	15.30	10.40	14.25	13.50
Tactics	+ SM	12.60	10.20	11.45	9.35	9.50	11.40
Tactics	+ SM	+ CC	11.45	9.90	10.65	8.85	8.55
Tactics	+ SM	+ CC	+ PW	9.65	9.50	9.15	7.45
Tactics	+ SM	+ CC	+ PW	+ IS	14.85	7.50	8.35
Tactics	+ SM	+ CC	+ PW	+ IS	+ KS	10.20	7.80
Tactics	+ SM	+ CC	+ PW	+ IS	+ KS	+ PS	10.70
= *Phoenix*							

Table 1: Accumulative Knowledge Result!

References

I.D. Kopec and M. Newborn, The Fourth World Computer Chess Championship, *Communications of the ACM 27,* 8(1984), 845-849.

. Jonathan Schaeffer, The Relative Importance of Knowledge, *ICCA Journal* 7, 3 (1984), 138-145.

. J.J. Gillogly, Performance Analysis of the Technology Chess Program, CMU-CS-78-189, Carnegie-Mellon University, 1978.

T.A. Marsland and P.G. Rushton, Mechanisms for Comparing Chess Programs, *ACM Annual Conference,* 1973, 202-205.

Ken Thompson, Computer Chess Strength, *Advances in Computer Chess 3,* (1982), 55-50, Pergammon Press.

E. Slater, Statistics for the Chess Computer and the Factor of Mobility, *Symposium on Information Theory,* Ministry of Supply, London, 1950, 150-152.

Program	Rating
Tactics	1110
+ Space & Mobility	1404
+ Center Control	1512
+ Pawn Weaknesses	1570
+ Incremental Scores	1556
+ King Safety	1750
+ Pawn Structure	1758
+ Planner	1786
= *Phoenix* (actual)	1840

Table 2: Effect on Rating of Additional Knowledge