7

Semi-autonomous acquisition of pattern-based knowledge

J. R. Quinlan†
Baser Department of Computer Science
University of Sydney

INTRODUCTION

This paper has three themes:

(1) The task of acquiring and organizing the knowledge on which to base an expert system is difficult.
(2) Inductive inference systems can be used to extract this knowledge from data.
(3) The knowledge so obtained is powerful enough to enable systems using it to compete handily with more conventional algorithm-based systems.

These themes are explored in the context of attempts to construct high-performance programs relevant to the chess endgame king-rook versus king-knight.

Most existing expert systems are based on knowledge obtained from a human expert. With reference to the family of geological expert systems being built at SRI, Gaschnig writes:

> Model development is a cooperative enterprise involving an exploration geologist who is an authority on the type of deposit being modelled, and a computer scientist who understands the operation of the PROSPECTOR system [1].

Feigenbaum, one of the pioneers of expert systems work and head of probably the world's largest group building such systems, puts it as follows:

> (The knowledge engineer) works intensively with an expert to acquire domain-specific knowledge and organise it for use by a program [2].

The expert is called upon to perform a most exacting task, with which he is also unfamiliar. He must set out the sources and methodologies of his own

† Present address: Rand Corporation, Santa Monica

expertise, and do so in such a way that it makes sense to a non-expert (the knowledge engineer) and can even be represented in a precise machine-usable form! Not surprisingly, this often turns out to be an onerous task with many false starts, so that Feigenbaum goes on to state that:

> The acquisition of domain knowledge (is) the bottleneck problem in the building of applications-oriented intelligent agents.

Inductive inference is a process of going from the particular to the general. We invoke this process ourselves each time we hypothesize some property shared by members of one set of things that differentiates them from everything else. At its most general level, an inductive inference system is capable of discovering regularities that can explain observed phenomena in some domain. Of course these regularities can be knowledge in a most compact form, just as the regularity $F = Ma$ embodies much of our knowledge of mechanics. Ability to discover regularities gives a new possible prescription for acquiring knowledge. The expert, instead of trying to specify the knowledge directly, guides an inductive inference system in its search for regularities in collections of examples drawn from his domain of expertise.

Once an expert system has been formulated, driven by knowledge obtained from inductive inference or otherwise, we can ask a number of questions about the quality of the system (and indirectly the knowledge on which it is based). Is it always accurate? How expensive is it to run compared to other systems, expert and otherwise? In the experiments to be described here, programs constructed via the inductive inference route were shown to be perfectly accurate, and to run up to two orders of magnitude faster than a commonly-used algorithm and five times faster than a knowledge-based system that the author constructed by hand.

This paper contains a brief survey of some existing inductive inference systems, and a more detailed examination of the particular system used for these experiments. There follows a discussion of the endgame king-rook versus king-knight and the results of experiments to construct programs for deciding the knight's side is lost in 2- or 3-ply.

INDUCTIVE INFERENCE: SOME EXAMPLES

The purpose here is to describe a few modern inductive inference systems, indicate the mechanisms they employ, and mention one or two notable successes of each. No attempt is made at completeness. For a more comprehensive treatment the reader is referred to survey papers [3, 4].

Meta-DENDRAL

Probably the most successful application of inductive inference techniques to the problem of knowledge acquisition is the Meta-DENDRAL program [5]. When an organic molecule is bombarded by high-energy particles it breaks into fragments (and some atoms may migrate between fragments). If the mass and

relative abundance of all the fragments can be found, an expert mass spectroscopist can identify a handful of possible structures of the original molecule. Meta-DENDRAL's original task was to discover the rules by which these structures could be deduced, or in other words to develop a theory of how molecules fragment. For each run the data consisted of a large number of known molecules together with their observed mass/abundance readings. The program first identified all possible places that the molecules could have broken in order to explain the observed fragments – only those breaks consistent with a 'weak' theory of what bonds can break were considered. Next a coarse search was made for possible rules to explain these breaks in terms of the local context of each broken bond. Finally the set of rules so obtained was refined, so that rules predicting breaks that did not occur were made more specific, and other rules were made more general if possible. The rules resulting from this three-phase process not only accurately reflected expert knowledge, but also included previously unknown rules. When the same approach was taken for nuclear magnetic resonance spectroscopy, once more new and useful rules were found. So successful was this work that the rules have been published as chemistry.

Meta-DENDRAL is an example of a special-purpose inductive system – it can only be used for the particular induction tasks for which it was designed. (It does however embody powerful kernel ideas, such as using a weak model to guide the discovery of a strong one, that have wide applicability.) General induction systems, on the other hand, are able to attempt problems of discovering regularities in domains where no semantic information is available to guide the inference. This class of systems is differentiated primarily by constraints on the form that discovered knowledge can take, and to a lesser extent by the way that search for this knowledge is carried out.

INDUCE

Michalski's INDUCE package [6] takes the description of a number of objects of different classes and constructs one or more generalized descriptions of each class. The descriptions of both objects and classes are couched in VL21, a language based on first-order logic but allowing typed variables. To find generalized descriptions of a class, the descriptions of all objects of that class are placed in one set (the positive instances) and all other objects into another set (the negative instances). A single description is then chosen from the positive set, and some of its atomic descriptors are built into more and more complex descriptions until a number are found that are sufficiently specific so that no object matching them can be in the negative instances. The best of them is selected as one of the descriptions of the class, the positive instances are reduced by removing all those that match this description, and the process repeated until all the original positive instances are covered by one or more of the generalizations. This system has also been applied to the problem of building expert systems, specifically for constructing a rule to test for a soybean disease. From the given descriptions of some hundreds of plants it was able to construct a generalized description of

diseased plants that was more accurate diagnostically then a human expert working from the same data. A precursor to the INDUCE system was also used in a similar series of experiments to those reported here, relating to the king-pawn versus king endgame [7].

THOTH-P

Vere's THOTH-P [8] is an example of a system that, while it is less generally applicable than INDUCE, can still tackle a range of tasks. Its data is a set of pairs of objects viewed as before-and-after snapshots, and it attempts to find the smallest number of relational productions that explain the changes. A relational production specifies that in a given context some stated properties are invalidated and new ones created, where the context and properties are again expressed in a language derived from logic. The method used to find these relational productions is an exhaustive search for maximal common generalizations which expands exponentially with the number of pairs of objects, and so cannot be applied to more than a small amount of data. (This exponential time problem is shared by other 'complete' systems such as SPROUTER [9]). Examples of its applications are finding the smallest number of primitive actions sufficient to explain a sequence of changes in the microcosm known as the 'blocks world', and discovering rules to change a restricted class of sentences from active to passive voice.

ID3

A fuller description of the general induction system that was used for the experiments reported here follows. In this case, the knowledge discovered is in the form of decision trees for differentiating objects of one class from another. Although this format is much more constrained than, for instance, the descriptions that INDUCE can generate, its simplicity is counter-balanced by its efficiency [10].

The basic algorithm on which ID3 is built is a relative of the Concept Learning System developed by Hunt in the '50s and documented in [11]. We start with a set of *instances*, each described in terms of a fixed number of *attributes*. Each attribute in turn has a small number of discrete possible *values*, and so an instance is specified by the values it takes for each attribute. To illustrate the idea, we could describe a person in terms of the attributes *height*, *colour of hair*, and *colour of eyes*. The attribute *height* might have possible (metric) values {1.30, 1.31, ..., 2.29, 2.30} while the attribute *colour of hair* might have possible values {dark, fair, red}. A particular individual might then be represented by the instance

$$height = 1.81,\ colour\ of\ hair = \text{dark},\ colour\ of\ eyes = \text{brown}\ .$$

With each of these given instances is associated a known class, which will here be either *plus* or *minus*. The task is to construct a decision tree that maps each instance to its correct class.

Suppose then that we have such a set C of instances. If C is empty then we cannot say much about it, merely recording it as null. If C contains one or more instances and they all belong to the same class, we can associate C with this class. Otherwise C contains two or more instances, some *plus* and some *minus*. We could then choose an attribute A with permissible values $A_1, A_2, \ldots, A_n$ say. Each member of C will have one of these values for A, so we can sort C into subsets $C_1, C_2, \ldots, C_n$ where C_1 contains those instances in C with value A_1 of A, C_2 contains those with value A_2 of A, and so on. Diagrammatically we can represent this as:

attribute A:
$A_1 \rightarrow C_1$
$A_2 \rightarrow C_2$
...
$A_n \rightarrow C_n$.

Now we have n sets of instances of the form C_i that we wish to relate to their class, so we can apply the same process to each of them. We keep going until each collection of instances is empty or all its members belong to the same class.

To clarify this process we will apply it to the simple problem from the king-rook king-knight endgame of determining whether the Black king can capture the White rook on its next move. There are two attributes:

- Black king is next to rook
- White king is next to rook

each of which has the possible values true (t) and false (f). The class will be *plus* if capture is possible, *minus* if it is not. The given set will contain all possible instances:

{*tt: minus, tf: plus, ft: minus, ff: minus*}

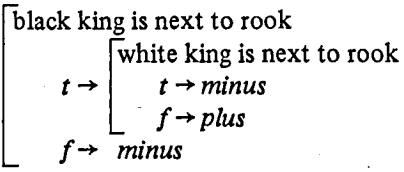
If the first attribute is selected we will have

black king is next to rook
$t \rightarrow$ {*tt: minus, tf: plus*}
$f \rightarrow$ {*ft: minus, ff: minus*}

The same process is next applied to the first (sub) collection, this time selecting the second attribute.

black king is next to rook
$t \rightarrow$ white king is next to rook
$t \rightarrow$ {*tt: minus*}
$f \rightarrow$ {*ft: plus*}
$f \rightarrow$ {*ft: minus, ff: minus*}

All subcollections now contain instances of only a single class; we can replace the collections by classes giving the decision tree



This rule is isomorphic to the program fragment

```
if black king is next to rook
then
    if white king is next to rook
    then capture is impossible
    else capture is possible
else capture is impossible
```

The above process does not specify how we should go about choosing the attribute A to test next. It is clear that any choice will eventually lead to a correct decision tree. An intelligent choice, however, will usually lead to a simple tree, while the straightforward algorithm 'choose the next attribute' will in general give rise to a monster more voluminous than the instances themselves.

In the original CLS the choice was made on a cost basis. Suppose we define a set of costs $\{P_i\}$ of measuring the ith attribute of some instance, and $\{Q_{jk}\}$ of misclassifying it as belonging to class j when it really is a member of class k. If we arbitrarily say that all instances in C are members of some class even when they are not we will incur a misclassification cost which depends on the instances in C and the class chosen. Let us denote by X_0 the minimum such cost over all classes. If we test some attribute A we will incur a measurement cost plus the sum of the total costs for each of the subcollections $C_1, \ldots, C_n$ resulting from testing that attribute – denote by X_1 the minimum sum over all attributes. The total cost of a rule for C is then the minimum of X_0 and X_1. This computation of total cost is recursive and can be prohibitively expensive if there are many attributes, but it can be approximated by a fixed-depth lookahead in much the same way that the true minimax function can be approximated by a fixed-ply search. The rule that results from this approach tends to have low or even minimal cost. Such as approach is particularly appropriate for applications like medical diagnosis where obtaining information has significant cost, and where certain classification errors are more acceptable than others.

Another line of attack (suggested to me by Peter Gacs of Stanford University) is based on information theory. A decision tree may be regarded as an information source that, given an instance, generates a message *plus* or *minus*, being the classification of that instance. If the probability of these messages is p^+ and p^- respectively the expected information content of the message is

$$-p^+ \log_2(p^+) - p^- \log_2(p^-) .$$

With a known set of instances we can approximate these probabilities by relative frequencies, i.e. p^+ becomes the proportion of instances in the set with class *plus*. So we will write $M(C)$ to denote this calculation of the expected information content of a message from a decision tree for a set C of instances, and define $M(\phi) = 0$.

Now consider as before the possible choice of A as the attribute to test next. The partial decision tree will look like

attribute A:

$A_1 \rightarrow C_1$
$A_2 \rightarrow C_2$
...
$A_n \rightarrow C_n$

and the new expected information content will be

$$B(C,A) = \sum_{i=1}^{n} (\text{probability that value of } A \text{ is } A_i) \times M(C_i)$$

where again we can replace the probabilities by relative frequencies. The suggested choice of attribute to test next is that which 'gains' the most information, i.e. for which

$$M(C) - B(C,A)$$

is maximal. When all the algebraic dust has settled, this comes down to selecting that attribute A (with values $A_1, A_2, \ldots, A_n$) which minimizes

$$\sum_{i=1}^{n} -n_i^+ \log_2 \frac{n_i^+}{n_i^+ + n_i^-} \; -n_i^- \log_2 \frac{n_i^-}{n_i^+ + n_i^-}$$

where n_i^+ denotes the number of *plus* instances in C that have value A_i of attribute A. This method of choosing the next attribute to test has been used now in a substantial number of different experiments, and does seem to give compact decision trees.

ID3 was specifically designed to deal with large numbers of instances. It uses the basic algorithm above to form a succession of (hopefully) more and more accurate rules until one is found that is satisfactory for all given instances. The basic paradigm is:

(1) Select at random a subset of the given instances (called the *window*).
(2) Repeat:

- Form a rule to explain the current window.
- Find the exceptions to this rule in the remaining instances.
- Form a new window from the current window and the exceptions to the rule generated from it,

until there are no exceptions to the rule.

Experiments (reported in detail in [12]) indicate that this process converges rapidly, and that, other things being equal, it enables correct rules to be discovered in time linearly proportional to the number of instances.

THE PROBLEMS ATTEMPTED

As mentioned in the introduction, this work has been concerned with subdomains of the chess endgame king-rook versus king-knight. Kopec & Niblett point out that this end-game provides quite a challenge even for masters. (The same paper is the source of many of the quantitative statements made here.) With one exception, each position in this end-game leads to a loss or draw for the knight's side, which we will take to be black.

The subdomains explored have been of the form *knight's side is lost in at most n-ply*, the main work revolving around the cases $n = 2$ and $n = 3$. This relation is defined as follows:

(1) A Black-to-move position is lost 0 ply if and only if

- The king is in checkmate, or
- The knight has been captured, the position is not stalemate, the White rook has not been captured and the Black king cannot retaliate by capturing it.

(2) A White-to-move position is lost in at most n ply (n odd) iff there is a White move giving a position that is lost in at most n-1 ply.

(3) A Black-to-move position is lost in at most n-ply (n even) iff all possible Black moves give positions that are lost in at most n-1 ply.

These definitions ingnore the repetition and 50-move rules of chess, but for small values of n are quite accurate. For brevity we will now omit the 'in at most' – 'lost n-ply' will be taken as shorthand for 'lost in at most n-ply'.

There are more than 11 million ways of placing the four pieces to form a legal Black-to-move position. The corresponding figure for White-to-move is more than 9 million. (The difference arises because, for instance, the White king cannot be in check in a Black-to-move position.) These counts, however, include many symmetric variants of essentially the same position, and when these are removed the numbers become approximately 1.8 million and 1.4 million respectively. About 69,000 of the 1.8 million Black-to-move positions are lost 2-ply while roughly 47,4 000 of the 1.4 million White-to-move positions are lost 3-ply.

Each position is described in terms of a set of attributes, which varied from experiment to experiment. These attributes are intended to bring out properties of a position that are relevant to its game-theoretic value. At present the system must be provided with the definition of these attributes, and the rules it finds are then decision trees relating the class of a position to its values of the given attributes. Two points are worth noting:

(1) It is possible for two or more distinct positions to have the same values for each attribute, and in fact the above large numbers of positions generally give rise to a much smaller number of distinct descriptions in terms of attribute values.

(2) If two positions of different classes have the same attribute values, it is not possible for a decision tree that uses only these values to differentiate between them. In such a situation the attributes are termed *inadequate*, and the remedy is to define some further attribute capable of distinguishing between the troublesome positions.

Finding small but adequate sets of attributes for the king-rook king-knight problems was a considerable task (at least for a chess novice like the author) – hence the 'semi-autonomous' in this paper's title.

THE 2-PLY EXPERIMENTS†

The first series of experiments was reported extensively in [14]. A set of seven problems was formulated, placing constraints on the positions analysed (e.g. restricting the Black king to a corner) and/or simplifying the meaning of 'lost' (e.g. ignoring stalemate). The attributes used initially were fairly low-level ones such as the distance between pieces in king-moves. As each problem was solved it served as a stepping-stone to the next.

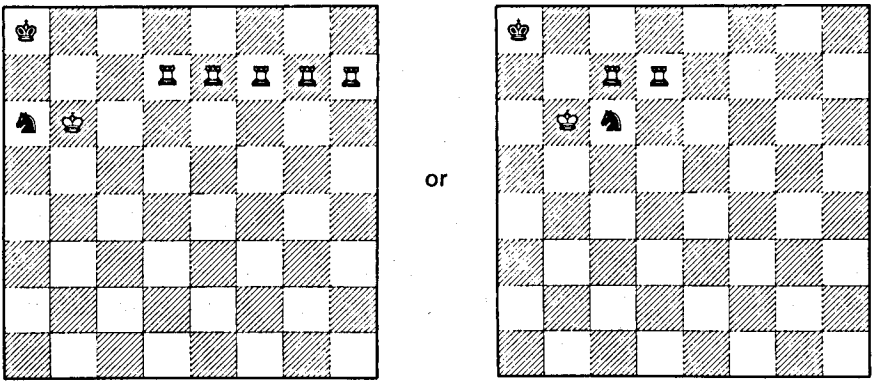
The final (unrestricted) problem was the full lost 2-ply task. The number of attributes had grown by then to 25, of which 21 were low-level ('Black king, knight and rook are in line') while 4 were decidedly more complex ('the only move that the Black king can make creates a mate threat'). Every possible Black-to-move position was described in terms of these attributes, which turned out to be almost adequate in the sense of the last section – eleven small sets of positions containing representatives of both classes could not be differentiated using the attributes. The 1.4 million positions gave rise to just under 30,000 distinct descriptions in terms of these attributes, and an implementation of ID3 in PASCAL running on a DEC KL-10 found a decision tree containing 334 nodes in 144 seconds.‡

The attributes were a rather motley collection, though, and some of them were expensive to compute. After considerable trial and error a quite different set of 23 binary-valued attributes was developed. These were all high-level, but

† These were carried out while I was visiting Stanford University. I am most grateful for the resources generously supplied by the Artificial Intelligence Laboratory and the Heuristic Programming Project.

‡ The earlier version of ID3 reported in the above paper used a different method of selecting which attribute to test next – it found a tree of 393 nodes in 394 seconds.

couched in terms only of broad patterns and operations on sets of positions. For example, one attribute was true if the given position was of the form



where the White rook was at one of the squares marked. Despite their smaller number, they turned out to be adequate. No two positions of different classes had the same value for all attributes. Even more surprisingly, the total space of 1.8 million positions collapsed into only 428 distinct instances. ID3 found a decision tree of 83 nodes for these instances in a couple of seconds. This decision tree was necessarily exact, since the instances it covered represented all possible positions.

The decision tree now gives a procedure for deciding whether a position is or is not lost 2-ply. Starting at the root, we find the value in the given position of the attribute tested at this node. Depending on this value we select one of the subtrees and continue until the selected subtree is a leaf. The given point is lost 2-ply if and only if the leaf is the class name *plus*. Notice that we may only have to evaluate a small subset of the attributes to classify any one position.

There are other ways of arriving at the same classification, and it is natural to compare them. The most obvious is *minimax search* which just interprets the definition of lost *n*-ply given in the previous section (with the usual alpha-beta cut-offs so that explorations that do not affect the classification are skipped). In the 2-ply case, the possible Black moves are examined to try to find one that results in a position that is not lost 1-ply. To determine whether a position is lost 1-ply, the possible white moves are tried in a a search for one that gives a lost 0-ply position, and so on.

Another classification method is *specialized search*, where we take into account additional information from this class of positions. For instance, to determine whether a position is lost 1-ply we need only examine White king or rook moves that capture the knight and White rook moves to the edge of the board (for a possible mate). This specialized search is really nothing more than an expert system that exploits domain knowledge – it is considerably harder to write and debug than minimax, but is more efficient.

Of course the simplest way to see if a position is lost 2-ply is to know beforehand all lost 2-ply positions and see if the given position is one of them. Several variants of this *look-up* are possible, but the one chosen here was to keep all lost 2-ply positions in memory, sorted so that they could be examined by a binary search. If a position could not be found among them then it was known to be not lost 2-ply.

PASCAL programs of roughly equal polish were prepared for each of these methods, and run on the same randomly-chosen collection of one thousand positions. They were cross-checked against each other to make sure that the classifications were the same in all cases†, and the average time on a DEC KL-10 to decide whether or not a position was lost 2-ply computed. The results in Fig. 1 raise some interesting points. First and foremost, the method using the second induction-generated tree is the fastest at 0.96 ms, edging out look-up at 1.12 ms. Secondly, even though the second collection of attributes was very different from the first, and gave rise to 70 times the compression, the performance of the first tree at 1.37 ms was not too dissimilar. Finally, if we measure the computation not by time alone but by the product of time and memory as suggested in [15], classification by the second decision tree found by ID3 is still the preferred method, rivalled this time by specialised search.

Classification method	CPU time (ms)	Memory required (X 1K words)	Time X memory
Minimax search	7.67	2.1	16.1
Specialized search	1.42	2.2	3.1
Look-up	1.12	67.7	75.8
Using first decision tree	1.37	4.3	5.9
Using second decision tree	0.96	2.5	2.4

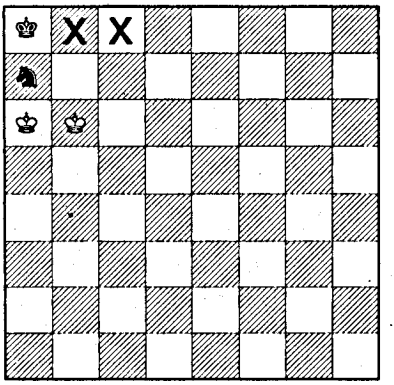
Fig. 1 – Comparison of classification methods for lost 2-ply.

THE 3-PLY EXPERIMENTS

The 2-ply case is perhaps special in that only a small proportion (less than 4%) of all possible positions are lost 2-ply. In the 3-ply case the corresponding figure is nearly 34%, and so the two classes are more evenly balanced.

† The thousand positions did not happen to include any of the very few for which the first set of attributes was inadequate.

About two man-months were required to find an adequate set of attributes for the 3-ply case. Of the final set of 49 binary attributes developed, 35 were concerned with patterns on the board such as



where the White king may occupy either of the indicated squares and the rook can move to some square other than those marked X in the same row as the black king. Four properties detected simple relationships like "White king is in check". The remaining ten dealt with relatively complex chess predicates such as "White rook is safe from reprisal 2-ply by the Black king if White king takes the knight or threatens to do so". Each of these ten predicates may be regarded as a small expert system in itself. Some are only approximate. The one above is defined as any of the following.

- The rook is more than two squares from the Black king.
- One blank square separates the rook from the Black king, but the rook is either next to or threatens the knight.
- The rook is next to the Black king, but either the knight is also next to it or there is a square that is next to the White king, knight and rook.

Others may contain bugs. However, it is interesting to note that the correctness of the final decision tree does not depend on the correctness of these subsytems used as attributes. So long as each attribute will always give the same value for the same position, and the collection of attributes is adequate to differentiate between positions of different classes, the decision tree produced by ID3 will be exact.

Even though some of them were rather untidy, the 49 attributes were adequate for this task, reducing the 1.4 million positions to 715 instances. ID3 was run on a CDC Cyber 72, and found a correct decision tree containing 177 nodes in 34 seconds.

Again a variety of classification techniques for lost 3-ply was tried. The minimax search was similar to the previous one, and the specialized search was built on the 2-ply specialized search using additional rules such as:

- To decide whether a position is lost (not more than) 3-ply it is advisable to check first if it is lost 1-ply.

- In establishing whether a position is lost exactly 3-ply, White moves that capture the knight need not be considered.

There was unfortunately insufficient space to store the nearly half a million lost 3-ply positions in memory, so look-up was not attempted.

One thousand White-to-move positions were generated randomly, and the average time taken to classify them by the different methods appears in Fig. 2. Minimax search is now much more expensive than the others, while the induction-generated tree is five times faster than specialized search. Unfortunately the PASCAL compiler used here did not give statistics on memory requirements, but it was found that the specialized search could execute with a field length about 40% less than required for the classification by decision tree. It would thus seem that Michie's computational measure would still rank the decision tree well ahead of search.

Classification method	CPU time (ms)
Minimax search	285.0
Specialized search	17.5
Using decision tree	3.4

Fig. 2 – Comparison of classification methods for lost 3-ply.

DISCUSSION

These lost 2- and 3-ply experiments demonstrate that expert systems built on knowledge inferred from data by an inductive system can match more conventional programs. In fact, if it proves possible to construct a lost 4-ply decision tree, it would be anticipated that the performance margin of the classification method using this tree over a specialized search program would once more increase dramatically.

These experiments were conducted over complete databases. In each case, every possible position was represented in the set of instances from which the decision tree was constructed. As a result, both decision trees were known to be exact. For some problems, however (such as an end-game with more pieces), the generation of the set of all possible positions may be computationally infeasible. This would not invalidate the technique, though, because experiments [12] indicate that a decision tree formed from only a small part of a collection of instances is accurate for a large proportion of the remainder. Typically, a rule formed from a randomly-chosen 5% of a large set of instances is also accurate for more than 75% of the rest. It could even be argued that this is the only genuine inductive inference, namely drawing conclusions from known instances that apply to those as yet unseen, and that working with complete databases is instead some form of information compression. At any rate, the decision tree produced

from incomplete databases will most likely be inexact and will have to be modified as exceptions are discovered, in much the same way as a complex program like an operating system is debugged as it is used.

In their present form, inductive inference systems are sufficiently powerful to extract high-quality knowledge from large numbers of instances, provided that the instances are described by appropriate attributes. Another side of induction (termed "constructive induction" by Michalski) is concerned with the much harder problem of developing good attributes from raw specifications. This problem will probably dominate inductive inference research in the eighties.

REFERENCES

[1] Duda, R., Gaschnig, J., and Hart, P., (1979). Model design in the PROSPECTOR consultant system for mineral exploration. *Expert Systems in the Micro Electronic Age*, pp. 153–167, (ed. Michie, D.). Edinburgh: Edinburgh University Press.

[2] Feigenbaum, E. A., (1977). The art of artificial intelligence 1: themes and case studies of knowledge engineering. *STAN-CS-77-621.* Stanford: Department of Computer Science, Stanford University.

[3] Mitchell, T. M., (1979). An analysis of generalization as a search problem. *Proc. 6th Intl. Joint Conf. on Artificial Intelligence, Tokyo*, pp. 577–582. Stanford: Department of Computer Science, Stanford University.

[4] Dietterich, T. G., and Michalski, R. S., (1979). Learning and generalization of characteristic descriptions: evaluation and comparative review of selected methods. *Proc. 6th Intl. Joint Conf. on Artificial Intelligence (Tokyo)*, pp. 223–231. Stanford: Department of Computer Science, Stanford University.

[5] Buchanan, B. G., and Mitchell, T. M., (1978). Model-directed learning of production rules. *Pattern Directed Inference Systems* (eds. Waterman, D., and Hayes-Roth, F.). New York and London: Academic Press.

[6] Michalski, R. S., (1978). Pattern recognition as knowledge-guided computer induction. *UIUCDCS-R-28-927.* Urbana-Champaign: Department of Computer Science, University of Illinois.

[7] Michalski, R. S., and Negri, P., (1977). An experiment on inductive learning in chess endgames. *Machine Intelligence 8*, pp. 175–185, (eds. Elcock, E. W., and Michie, D.). Chichester: Ellis Horwood.

[8] Vere, S. A., (1978). Inductive learning of relational productions. In *Pattern Directed Inference Systems*, (eds. Waterman, D. A., and Hayes-Roth, F.). New York and London: Academic Press.

[9] Hayes-Roth, F., and McDermott, J., (1977). Knowledge acquisition from structural descriptions. *Proc. 5th Intl. Joint Conf. on Artificial Intelligence (Cambridge, Mass.)*, pp. 356–362. Pittsburgh: Department of Computer Science, Carnegie-Mellon University

[10] Cohen, B. L., (1978). A powerful and efficient structural pattern recognition system. *Artificial Intelligence*, **9**, 223–225.

[11] Hunt, E. B., Marin, J., and Stone, P., (1966). *Experiments in Induction.* New York and London: Academic Press.

[12] Quinlan, J. R., (1979). Induction over large databases. *HPP-79-14.* Stanford: Heuristic Programming Project, Stanford University.

[13] Kopec, D., and Niblett, T. B., (1980). How hard is the play of the King-Rook King-Knight ending? *Advances in Computer Chess 2*, pp. 57–73. (ed. Clarke, M. R. B.). Edinburgh: Edinburgh University Press.

[14] Quinlan, J. R., (1979). Discovering rules by induction from large collections of examples. *Expert Systems in the Micro Electronic Age*, pp. 168–201. (ed. Michie, D.). Edinburgh: Edinburgh University Press.

[15] Michie, D., (1977). A theory of advice. *Machine Intelligence 8*, pp. 151–168 (eds. Elcock, E. W., and Michie, D.). Chichester: Ellis Horwood.