Stanford Heuristic Programming Project
Memo HPP-79-14

May 1979

Computer Science **Department**
Report No, STAN-CS-79-739

INDUCTION OVER LARGE DATA BASES

by.

J. R. Quinlan

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Induction Over Large Data Bases

J. R. Quinlan
Basser Department of Computer Science
University of Sydney

Abstract: Techniques for discovering rules by induction from large collections of instances **are developed.** *These* **are based** *on* an *iterative scheme for dividing the instances into* **two** *sets, only one of which needs to be randomly accessible, These techniques* **have** *made It possible to discover complex rules from data* bases *containing many thousands of instances. Results of* **several** *experiments using them* **are** *reported,*

Keywords: Induction, Inference, pattern recognition, rule **formation,** *concept learning, decision trees.*

Introduction

Since the early work with Perceptrons the study of learning systems has had an Important place in artificial Intelligence, The term Yearning' of course covers a range of behavior. At one extreme there is the adjustment of numerical parameters that characterises for instance most of spatial pattern recognition [Nilsson 1965]. At the other we now have programs such as AM [Lenat 1977] that not only acquire new structures but also the conceptual framework on which to hang them. Somewhere between iies the area known as *Induction.* It has two general characteristics: its aim is to discover structured information about some collection of entities, and the methodology employed is the analysis of many examples of the *genre* called *Instances.* Thus learning programs such as [Quinlan 1975] are not Induction systems because the information gleaned from instances has little structure, Neither are those such as TIERESIAS (Davis 19773 because, although they find or modify structures, they do so by metalevel interaction with an expert rather than by the examination of instances.

As with most areas In Al, Induction systems can be divided into those that use only general methods and so are applicable to any problem, and those designed for specific tasks. Although it is not intended to attempt a survey here, a short discussion of a couple of existing systems should highlight important concepts, Much more substantial pointers Into the literature may be found in [Hayes-Roth and McDermott 1977], [Buchanan et al, 1977] and [Michalski 1978).

The work reported here was performed while the author was visiting the Computer Science Department of Stanford University, and using facilities made available there by the Artificial Intelligence Laboratory and the Heuristic Programming Project,

The earliest programs dating from the late fifties **were the series of Concept Learning Systems (CLSs),** an extensive exploration of which may be found in [Hunt et al, 1 **966].** They were general-purpose programs although they found particular application in finding rules for pattern classification. Each instance presented to them is described as **a** vector of discrete values for **a** fixed number of attributes or properties, and ail **instances** must **have** this **identi**cat format. The output is **a** decision tree relating one of the attributes (normally **a** class) to the values of the others. This tree is constructed top-down, starting with all Instances In a single set **and** seeking successively finer partitions of this set until **each** block of the partition contains only instances of a single class. More details of this approach will be given later.

A more recent system in the **general-purpose** category is INDUCE **[Michalski** 1878). Here **each instance** is represented as a series of assignments of properties; **for example the fragment**

part: = ***P*1, *color*(*P*1):** = *blue,* part: **= *P*2, *color*(*P*2): = *yellow***

would indicate that the object being described has, among other things, **a** blue part and a yellow part. instances need not ail have the same number of components or properties, so there is quite **a** lot more denotational freedom than provided by the rigid vector of CLS. The output from INDUCE is **a** collection of decision rules couched in **a** quantified multi-valued logic, and may employ descriptors that appear nowhere in the specification of the instances such as

if there ***are*** *3 or more red* parts *then* ***It's*** a *fire engine*

Unlike CLS, INDUCE works bottom-up; starting from a collection of positive instances of **a** class and one of negative (counter-)exampies it attempts to build successively more complex descriptors that cover the positive but not the negative collections. Although this system does Indeed seem promising, **only** partial impiementations of it have been developed.

There are a few systems of intermediate generality, limited either by the sort of instances that can be **analysed** or the relations discovered. (For instance Thoth-p **[Vere** 1978) is designed to study ‘before and after’ or sequences of snapshots to find the actions responsible for the changes. The actions are represented by relational productions containing simple sets of first-order formulae to represent context, relations destroyed and **relations created,)** **However** most other induction programs are strongly related **to some task** area, The most widely-known of these must surely be **Meta-DENDRAL** [Buchanan et **al.1978]** which discovers rules for identifying organic chemicals from readings of an experimental device (originally **a mass** spectrometer), The instances it analyses are triples consisting of the graph of **a** chemical compound, a fragment mass and its relative abundance. The rules sought are predictions of how chemical bonds **will** break and atoms migrate when irradiated with high-energy particles, In forming trial rules this system is guided by a weak model of what can and cannot happen in the instrument; the final rules represent **a** much stronger model. This work is of interest on at **least** three counts. The use of a weak model **as a** sort of plausible move **generator is patently**

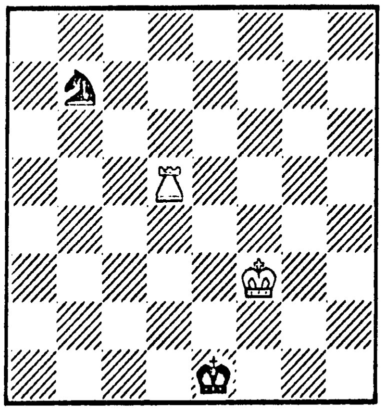


Figure 1: A counterexample

a good idea, Sets of rules that it produced have been published as chemistry, attesting to their power, Finally, it seems to be one of the few systems that has demonstrated en ability to cope with large volumes of data, which is the particular concern of this paper.

During his visit to Stanford in the fail of 1878, Donald Michie discussed the following chess problem. We consider an endgame situation in which the pieces have been reduced to a black knight, a white rook and the two kings, and where it is black's turn to move. We wish to know whether black is safe for at least one more white move. (The complete endgame situation is discussed fully in [Michie et al.1978].) Safety here has to do with keeping the game alive a little longer, if black can capture the rook or achieve stalemate then the game is drawn, the best possible outcome from black's point of view. A more exact specification is

- If the given situation is mate then black is not safe; if it is stalemate black is safe.
- Otherwise, black is not safe 2-ply if he cannot capture the rook and no matter what move he makes white can either mate him or capture the knight without stalemate or leaving the rook *en prise.*

Naturally this question can be resolved easily by search. The task however was to answer it in terms only of some static description of the initial situation. Simple tabulation of ail cases was ruled out because, even after symmetry has been taken into account, there are nearly two million possible configurations; the description of a situation had thus to characterise rather than define it. At least four people attempted the problem by hand, but ail 'solutions' turned out to contain errors resulting from plausible but unjustified assumptions. For instance it seems reasonable to assert that black is safe at least in those cases where neither the black king nor black knight are threatened, but Figure 1 shows a counterexample. it is not surprising then that attempts to discover a rule using an induction system met with considerable difficulty.

in the course of studying this and related problems, certain techniques were developed that seemed to have more general **applicability.** These **are** introduced in the next section, A particular induction task and algorithm are specified and two succeeding sections discuss in some detail the experiments carried out. These and additional results **are assessed in the** conclusion.

General Description of **the** Approach

The essence of the induction task is discovery, The data for the task is **a** collection of *instances* or descriptions of some set of entities in terms of their properties. A rule is some method of *explaining* an instance by establishing some relationship between these properties. The induction task is to discover a rule adequate to explain each instance in the data. For example, one common type of induction task is the discovery of a rule for classifying patterns. The data is usuaiiy referred to as *a training* set, and one property of each instance is its **class** membership. The induction task in this case is to discover a rule that relates the class of **each** item in the given training set to Its other properties, and hopefully also predicts the class of many items in **the entire** description space from which the training set **was** derived.

Several systems capable of discovering such rules to explain **a** set $\mathfrak{D}$ of instances were cited in the introduction. Their workings have little in common, but it seems safe to hypothesize that any system for this task must examine many of the instances in $\mathfrak{D}$ many times, and that the order in which instances are referenced is not fixed. This is fine so long **as the size of** $\mathfrak{D}$ **does** not preclude its storage in fast random-access memory, Many training sets **of** interest, however, are too large to store in this way, and the performance of any induction system could be expected to wilt if each reference to an instance required a disk operation or some such.

One **way** around this obstacle would be to select a manageable subset of $\mathfrak{D}$, form **a** rule to explain this subset, and hope that the rule is also adequate to explain $\mathfrak{D}$. As mentioned above, this is **a** common practice in pattern recognition, but commonsense (backed by **a few** experiments) indicates that, as the rule necessary to explain $\mathfrak{D}$ becomes more complex and the training set becomes **a** smaller subset of $\mathfrak{D}$, the probability of **a** rule derived to explain the subset also being adequate for all of $\mathfrak{D}$ approaches zero. As an iiiustration, **consider** the problem of discovering the rules of a two-person game from examining **a** random sample of **moves** in context. For a game such as tic-tat-toe quite a small collection of moves would probably suffice; in the case of chess the sample would have to be enormous before there was near certainty that each possible move pattern was represented In it, as **a** consequence both of the number of such patterns and the complexity of the context affecting their **legality,**

Another method is to break $\mathfrak{D}$ into chunks, each small enough to fit in primary .**memory,** and to process the chunks in a fixed sequence, This is also effective, but it requires **a more** complex algorithm and **may** lose advantages that could have arisen from the juxtaposition of certain **instances** which may now be spread over several chunks, **As an analogy, if** $\mathcal{A}$ **appears** in one chunk and $\mathcal{A} \Rightarrow \mathcal{B}$ in another, the task of discovering $\mathcal{B}$ becomes more taxing than if both the above facts had been found together,

Hunt and various co-workers [Hunt et al. 1966, Diehr and Hunt 1968] have also addressed this problem, though from a somewhat different perspective. Suppose that memory is limited to slots for some fixed number of instances. One method they proposed was to scan the data base in cycles, each instance being inspected once every cycle. The instance was read into a randomly chosen slot (thus destroying the record of any instance previously occupying that slot) and, if the current rule was not adequate to explain this instance, a new rule was immediately developed from only those instances currently occupying the slots. Thus many rules could be generated in the course of a single cycle, and many cycles could be necessary to yield a sufficiently correct rule (since this work did not require that the final rule be exact). A more sophisticated development saved memory space by storing incomplete instances. if the current rule explained an instance correctly, then only those of its properties used by the current rule were stored for that instance; if the current rule was incorrect then the entire instance was saved, Thus the available memory could at any time contain a mixture of partially and completely specified instances. This approach was able to produce quite accurate rules using very little memory (e.g. 95% accuracy using memory sufficient for about 5% of the data base), However the sample problems studied were extremely simple — the data base typically contained 266 instances, each described by four attributes, and a rule for determining the class to which an instance belonged needed to test only one or two of the attributes. This technique does not seem to have been tested on a problem of more realistic complexity.

This paper proposes another approach that does not require changes to the inductive algorithm being used, but instead attempts to separate out a distinguished subset of $\mathcal{D}$. The whole of $\mathcal{D}$ is presumed to be able to be accessed sequentially, such as by being held on some secondary storage medium. Note that, using buffering methodologies, it Is still possible to examine the elements of $\mathcal{D}$ with an average access time very close to that of primary memory, but **only** if we examine a large number of them and in a fixed sequence, The special subset, on the other hand, is distinguished by our ability to reference Individual Instances in it randomly and quickly; it is thus envisaged that this much of $\mathcal{D}$ will fit into primary memory. The focus of this paper is the development of iterative techniques for selecting from $\mathcal{D}$ the subset that can be readily manipulated, At any time the inductive algorithm will be able to 'see' the total data base only through this subset, which is referred to as a *window.*

The idea underlying the scheme is as follows. Let $\mathcal{W} \subseteq \mathcal{D}$ be a window into $\mathcal{D}$. Whatever induction system we are using should be able to produce a rule $\mathcal{R}$ that explains $\mathcal{W}$—if not then it is hardly likely to be able to find one to explain $\mathcal{D}$. Unless this rule $\mathcal{R}$ is a case-by-case tabulation of $\mathcal{W}$ (which most people would not regard as an explanation of $\mathcal{W}$), it will contain structures that also explain instances not in $\mathcal{W}$, and in particular some instances in $\mathcal{D} - \mathcal{W}$. in general $\mathcal{R}$ will not be sufficient to explain *all* of $\mathcal{D}$; $\mathcal{R}$ applied to $\mathcal{D} - \mathcal{W}$ wilt partition it into a set of instances corroborating $\mathcal{R}$ and a set of exceptions for which $\mathcal{R}$ offers no explanation or for which its explanation is incorrect. Now if $\mathscr{R}$ is viewed as an erroneous explanation mechanism that must be debugged, then these exceptions (i.e. the bugs) would seem to be more important than the corroborating instances, They may be discovered, of course, by examining all instances of $\mathscr{D}$ one at a time in some arbitrary order; as noted above this can be performed out of secondary memory without a significant increase in the average access time. The suggestion then is to start by selecting $\mathscr{W}$ at random from $\mathscr{D}$, and to form a new window from the old window and the exceptions to the rule formed to explain that window, The old rule is then discarded, a new rule is found to explain the new window, and the process repeats until a rule is discovered that has no exceptions and so explains all of $\mathscr{D}$. Note that only one rule is developed on each iteration, and that the final rule is exact (although the process could of course be halted as soon as the number of exceptions fell below some threshhold),

The most obvious way of forming a new window from the previous one and a set of exceptions would be simply to merge them. However these exceptions may be redundant in the same sense that the original data base $\mathscr{D}$ is redundant. The window would then grow rapidly, leading to the same sort of storage troubles that the whole approach is supposed to overcome. instead the *first method* places a limit (called the *exceptions limit*) on the number of exceptions that can be added to the window at each iteration. ideally this limit should be neither too high (in which case the window grows fat) nor too low (when an inordinate number of iterations would be required because the window changes so slowly). This question is explored later.

The first method above still suffers from the disadvantage that the size of the window increases at each iteration, it could happen that at some stage the window occupies ail available storage and the process must then terminate as no additional exceptions can be incorporated, The *second* method forms a new window by blending instances from the old window with exceptions in such a way that the window size remains constant. An attempt is made to ensure that at least one instance corresponding to each subcomponent of the old rule is included in the new window so that sections of the old rule that may be correct are not forgotten, Here also it seems sensible to restrict the number of exceptions added at each iteration, not to stop the window growing too rapidly, but rather to prevent the old window being swamped, and thus essentially discarded, each iteration,

Two properties of both methods are worth noting. In each case the window changes only slightly when there are few exceptions to the rule derived from it. As the rule developed from a window approaches a correct rule for all of the data base 9, the next rule is thereby constrained to be similar. On the other hand when there are many exceptions, the window and the rule generated from it can change more rapidly. The rate of change of the rule thus agrees with an intuitive approach to hill-climbing. However the methods are heuristic in nature and neither can guarantee convergence on a correct rule. The first will stop when the window size reaches the bounds of the storage available for it, so at least it will always terminate. The second, though, may search forever attempting to construct a window of fixed size from which a correct rule for ail of $\mathscr{D}$ can be induced, if there is no such subset of $\mathscr{D}$—for instance if the number of subcomponents of **any** correct rule exceeds the number of **instances allowed** in the window-then the search is doomed to failure.

The Problem Attempted

The **preceeding** section outlined an iterative technique for discovering **a** rule to explain **a data base,** and two methods of forming a new window **as** part of this technique. The problem chosen as **a** vehicle for studying them had to have a number of characteristics, First, **it had** to be real, it would have been easy to create an **artificial** problem by choosing a set of 'attributes' and arbitrarily assigning subregions of the space so created to classes. This would inevitably build into the task preconceptions of what such a task should look like, Secondly, a substantial data base was necessary because the techniques are primarily concerned with problems of storage that do not arise with small training sets. The rule sought had to be non-trivial, because difficulty could be expected to magnify the difference between **a** sound **and** an unsound approach, But in spite of ail these the task had to be of **manageable** proportions. it was anticipated that the number of trials necessary to evaluate the approach would be in the hundreds, and although the computer **facilities** generously made **available were substantial** they were not inexhaustible.

The task settled on was related to the chess problem posed by **Donald** Michie, but **scaled** down in complexity. Again there are only four pieces on the board (black king and knight, white king and rook) with black's turn to move. The question asked is this:

> ***Given:*** *a* ***position (neither*** stalemate *nor checkmate) In which the black* ***knight*** *Is pinned or skewered by the rook, or the rook has achieved* **a** ***linear*** *fork of the black* ***knight*** *and king. After black's move, can white Immediately* capture *the knight without either* ***leaving*** *the rook en* ***prise*** *or* causing *stalemate?*

Two examples are shown in Figure 2. in the left-hand one the black king must move and, whatever he does, the rook can then capture the knight so the knight is *lost.* The right-hand case seems simiiar, but this time the black king can move to the corner. if he does so the rook cannot capture the knight without causing stalemate, so the knight is *safe.* As these *examples show, determining whether the knight is safe or lost two-ply under these conditions stilt requires care, Note that the original problem has been restricted in two ways, The Initial position Is **a** pin, (linear) fork or skewer instead of any arbitrary configuration of the pieces; this reduces the size of the data base. Secondly, we are ignoring the possibility that white can mate on his next move without capturing the knight; this reduces the complexity of the rule that must be discovered.

Fourteen quasi-geometric attributes relevant to this task were defined. Three **examples** should give their flavor:

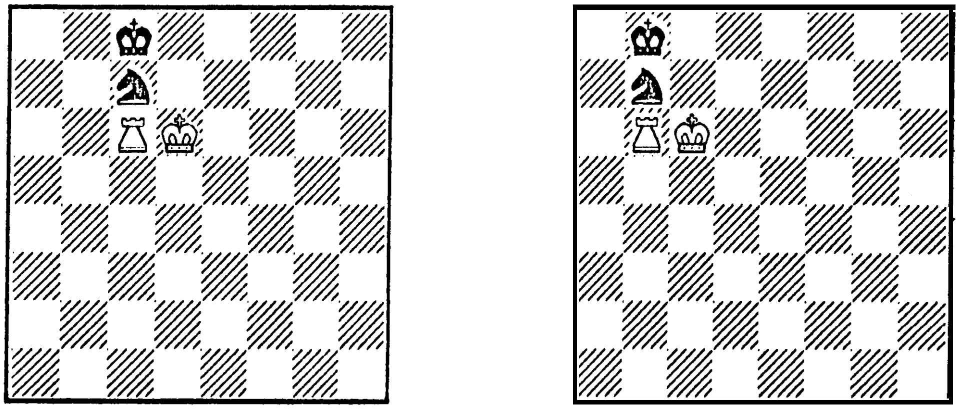


Figure 2: Two examples of initial positions

- The distance in king moves from the black king to the white rook, with possible values **1,** 2 and more than 2. If this distance is 1 or 2 there is a possibility **that the** black king can move to threaten the rook.
- Whether or not the black knight occupies a square one **away** from **a** corner **and on a** diagonal. As Figure 2 illustrates, this can be important in deciding whether or not black can threaten stalemate.
- Whether or not the black king can move so that it is next to the knight, and thus defending it, This is the least geometric of ail the attributes, but it can still be thought of in terms of set operations rather than search. if we denote the set of posltions

  adjacent to the black king by *bk*
  adjacent to the white king by ***wk***
  adjacent to the black knight by ***bn***
  in the white **rook's** rank or file by ***wr***
  ditto but with the knight between them and the rook **by *shadow***

  then this attribute can be expressed as

  $bk \cap bn \cap (-wk) \cap ((-wr) \cup shadow)$ is non-empty

Nine of these attributes had two permissible values, five of them had three.

Ail possible initial chess positions as above were generated and the values of the fourteen attributes plus the **class** ***(lost*** or safe) determined. These attributes do not define **a** position uniquely: many different positions can map into a single vector of attribute values, so the number of distinct vectors is a small proportion of the number of possible positions. Nor are they independent: most of the $2^9 \times 3^5$ points in the attribute space do not correspond to any position. in this **case** it was found that

- No two positions with the same vector of attribute values belonged to different **classes,** so the attributes are adequate for this task. **Had** this not been the **case,** there would have been positions for which it would not have been possible to tell whether the knight was *lost* or *safe* from the values of the attributes alone, and so the task of explaining the data **base** would **have** been impossible.
- There are 1,987 distinct vectors.

The data base (hereafter referred to as **pf&s) was** taken **as** this entire collection of nearly two thousand instances, Since it is complete, any rule that explains it gives **a** precise answer to the question posed above. The complexity of an exact rule couched in terms of these attributes is brought out by the Appendix, which gives **a** full specification of the attributes and the simplest known correct rule in the form of a decision tree.

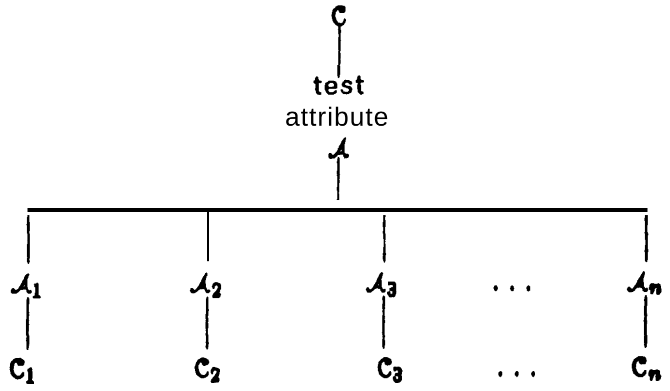
it is important to reaiise that the game of chess has no central role in this evaluation, but merely serves **as** the source of a non-trivial task that can readily be understood by human beings. Neither the induction algorithm nor the iterative technique being investigated have anything pertaining to chess associated with them. As far **as** the programs are concerned, this is **a** task of inducing a complex class-prediction rule from **a data base** of two **thousand** elements described by fourteen other attributes. Any task of similar dimensions would be handled similarly, whether it arose in the context of anaiysing bubble-chamber photographs or picking the winner of the Melbourne Cup.

The induction Algorithm Used

The iterative technique proposed earlier is clearly not tied to any one induction algorithm. its only interaction with the latter is to provide a subcollection of instances and to receive back a **rule** adequate to explain the subcollection. It should be possible to replace the algorithm used in this set of experiments **with** another and obtain comparable **results.** in fact the algorithm used here is **a** simple and unsophisticated relative of Hunt's CLS. The virtues of our version lie in the direction of compactness and efficiency rather than **'intelligence',** so if anything **a** more advanced algorithm should give better results.

CLS-like systems require that each instance be specified in the same fairly rigid format. We imagine some fixed list of attributes each having a (usually small) finite number of permissible values. For example color of eyes might be an attribute with permissible values blue, *green, brown, black* and *other. A* rule will relate one of these attributes to values of the others, The attribute so distinguished is normally called a class from the fact that this approach was developed in **a** pattern-recognition context,

Suppose we have a collection **C** of instances whose class we wish to relate to the other attributes, Three **cases** arise, if C is empty then we can offer no explanation of it, if ail members of C are of the **same** class we can relate them to this class without further testing. Otherwise we select some attribute $\mathcal{A}$ whose permissible values **are** $\mathcal{A}_1, \mathcal{A}_2, \ldots . \mathcal{A}_n$ (say) and partition C into subcoiieotions $C_1, C_2, \ldots C_n$ where C_i contains the members of C that have value $\mathcal{A}_i$ of $\mathcal{A}$. Graphically we have a structure



where each branch corresponding to a value of $\mathcal{A}$ contains a subcollection of instances, We then do the obvious recursive thing of applying the same process to each C_i In turn. The result is **a** tree, Each internal node causes us to branch on the value of some attribute, Each leaf is either ***null*** because no instances in the training set correspond to it, or contains **a** collection of instances of **a** single class which we will call the assigned class of that leaf, This tree is itself the rule that we seek, because it maps any instance to a leaf which either has an assigned class or is null.

The trick building this tree is in deciding at each stage which attribute $\mathcal{A}$ to branch on. Certain attributes can immediateiy be **exclued** from consideration. if ail but one of the C_i**'s** are empty then it would be pointless to select this attribute; this filter automatically prevents testing the same attribute twice. There is still quite a scope for choice, which can be made in many ways. As originally suggested by Hunt, we could use **a** system of costs: measurement costs for determining the value of an attribute, and **misclassification** costs estimated by **a** type of lookahead procedure. Using this approach we would select that attribute for which the sum of these costs **was** minimal, and so hope to build a minimum-cost rule. Alternatively we could use task-specific information to reject or suggest possible attributes in context, For example we could reject as unlikely the testing of whether the knight could flee before seeing whether it could capture the rook, or we might suggest that it is a good idea to start by finding whether the king is in check, Such a knowledge-based approach would use plausibility to guide the rule construction in much the same way that **Meta-DENDRAL** uses a weak model of its task domain to guide the construction of a strong one, and would tend to produce **a** rule that a human being might find easy to understand.

For these experiments, though, we used a much more straightforward task-independent heuristic aimed at giving a simple rule. (The choice of simplicity as the goal was made **because** such a rule is more likely to be general, and so cover instances not included in the window from which it was formed.) Suppose as above that we are considering attribute $\mathcal{A}$ **as** the next test.

If we select it, we will be left with the residual task of building trees for each of the C_i and we would like some predictor for the complexity of these subtrees. Each C_i is a collection of some number s_i of Instances of class *safe* and l_i of class *lost*. If either of these numbers Is zero then the tree for the branch containing C_i will be a leaf. If the number In either class Is small then we would expect that it would not take many more tests to separate these instances from the rest. After some trial and error we **found** that an estimate of the complexity of the tree required for C_i as

$$\sqrt{minimum(s_i,l_i)}$$

appears to work quite well, The total complexity of all residual **subtrees** If $\mathcal{A}$ Is selected is then the sum of these measures over the collections C_i. At any stage, the attribute selected as the next node on the tree is one that minimizes this estimated total residual complexity, and so the selection method is akin to a I-step lookahead. (Some additional experiments **are** currently being performed using an information-theoretic model of complexity.)

This Induction algorithm and the iterative technique that uses it exist as a collection of programs written In a dialect of PASCAL for the DEC **10[Kisicki** and **Nagel1976].** All execution times appearing In **this** report are for a KL **10.** The programs themselves were written wlth **an** eye to their adaptation for other problems; copies are available from the author.

Results Using the First Method

Recall that the first technique Involved selecting a subset of $\mathfrak{D}$ called the window, **and** repeatedly adding to it exceptions to the rule developed to explain it, with the restriction that no more than some fixed number of exceptions can be added **at** each iteration. The two Important variables here are the size of the initial window and the exceptions limit. If these numbers are very small then many iterations will be required to generate **a** correct rule; if too large, then not only will the window grow rapidly, but the time taken by the inductive algorithm to find **a** rule adequate to explain the window at each iteration **will** also Increase. Two examples with the **pf &s** data base should illustrate this. A run was made with the initial window containing **40** instances and the number of exceptions added at each iteration **limited** to 20 **(i.e.** 2% and 1% of the data base respectively). Ten iterations were required to generate **a** rule **correct** for all of $\mathfrak{D}$. Each was fairly short since the window was small; the total time required was 6.4 seconds and the **final** window size was 106 instances, In contrast, when the initial window **contained** 400 Instances and up to 200 exceptions could be added at each iteration (20% and 10% **respectively),** only four iterations were needed. The total time however was still 4.1 seconds due to the larger initial window size, As a consequence of this larger **initial** window there were fewer exceptions at each iteration and the final window contained 460 instances, More details of these runs appear in Table 1.

A better choice for the two parameters would seem to lie somewhere between these extremes. A number of runs was made with the initial window **size** ranging from 60 **to 360 and** the exceptions limit varying from 26 to 126. Each run was repeated 10 times with different random choices of both the initial window and the exceptions added at each iteration (when the number of the latter exceeded the limit). As before, three measurements are important:

The total time taken to find a correct rule; this gives a direct comparison of the suitability of different combinations of the initial window size and the exceptions limit.

The number of iterations required; this gives **a feel for the rate of convergence of** the process.

The final window size; this demonstrates the reduction in random access storage that may be achieved using the scheme.

The averages of these three measures over each group of ten runs Is shown In Tables 2a, 2b and 2c. They indicate that, for intermediate vaiues of the initial window size and exceptions limit, the first method exhibits quite consistent behavior. A correct rule Is typically discovered In about 4 Iterations and 3-3.5 seconds, with the final window containing 14% -17% of the total data base. This consistency is quite important to the usefulness of the approach. If it had turned out that performance was very sensitive to the choice of values for the initial window size and/or the exceptions limit, then using this method in practice would entail many trials to find a happy combination of values. The substantial 'floor' covering most Intermediate values of the tables is reassuring in that, If it carries over to other problems, **a** fairly arbitrary choice of these parameters is likely to yield acceptable results.

Given the complexity of the rule in the appendix, taking a little over three seconds to find It seems good, and doing so in about four iterations attests to the rapid convergence of the scheme. But the primary justification for the approach is that It requires less fast memory, In this case about one sixth of what would be needed to store the whole data base. To place this figure in context (at least with respect to the induction algorithm being used), we tried to produce a correct rule directly from large subsets of the Instances. Although more than 100 such subsets containing from 26% to 60% of the total data base were tried, In no case was a rule found that was adequate for ail the data. This contrasts with the performance of the iterative approach, which is able to find **a** correct rule **each** time using considerably less space.

Another result using this method may be of interest. If the Initial window size and exceptions limit are set very low, a correct rule is (eventually) obtained with **a** very compact final window. For example, one run with both of these parameters set to 1 found a correct rule with a final window consisting of only 88 instances, or about 4% of the total data base. This final wlndow represents in some sense a distillation of all the pf&s data, containing as It does all the 'special' cases and enough of the ordinary ones to indicate a correct rule. So this technique may also be an appropriate mechanism for sifting large data bases so as to build **a** tutorlal collection of Important cases.

Results With the Second Method

The second method differs from the first in the way that the new window is formed **at each** iteration, The rules governing its composition are:

- The window size remains fixed.
- At least one instance corresponding to each subcomponent of the rule developed from the old window also appears in the new one. In our case the rule Is **a** decision tree, each subcomponent of which Is Identified with **a** leaf. As before, some **leaves** are null In the sense that no instances map to them. So **each** non-null leaf In the old rule Is represented In the new window by one or more of the Instances **that mapped** to that leaf.
- No more than **a** fixed number of exceptions **can become part of the new window,** Some preliminary experiments were carried out varying this **limit** from 33% to 60% of the (fixed) window size; results were quite uniform, so the higher percentage was chosen.

The number of exceptions incorporated **into** the new window **was** then the minimum of

the number of exceptions to the old rule,

half the fixed window size, and

the window **size** less the number of non-null leaves in the old rule

and, as before, a random subset of this size was chosen from the exceptions If there were too many to Include them all. The instances from the old rule were **also** selected randomly; first, one corresponding to each non-null leaf in the old rule, and then an appropriately sized subset of the remainder,

The reason for developing this method was to ensure that the window would not expand to such an extent that it overran the storage available for it. Consider for example the results with the first method where the initial window contained 160 instances and the exceptions **limit** was 76, The average final window size over the ten runs is shown in Table **2c as** 297, but in five of those ten cases the final window contained more than 300 instances (the maximum was 343). If the total space available for storing the window had been sufficient for only 300 instances, the first method using these parameter values would have been successful **in** developing **a** correct rule in only half the trials-the others would have had **to be abandoned** because of insufficient space, **In** contrast, if the second method were used with **a** window size of 300 instances then it would always succeed, because there do exist collections of 300 instances from which It is possible to discover a correct rule.

This second method may be viewed then as containing an element of insurance, and **as** such should be expected to incur some cost in terms of increased times to find **a** correct rule.

As before, a number of dlfferent values of the only parameter (window size) was tried with ten runs each; the results are summarized in Table 3. This shows that any additional cost Is slight, as a typical run here requires about 3.5 seconds and 4 iterations. Once again the performance of the iterative scheme Is quite stable for intermediate values (i.e. window sizes around 16% of the total data base), but falls off rather sharply as the window size is reduced below a comfortable level. The reason for these increased times would seem to be that the number of subsets of a given size from which a correct rule can be induced declines rapidly with this size, and so the number of iterations necessary to find one of them Increases. On the other end of the scale, however, the performance is predictable-the slow increase in time taken represents the roughly linear additional effort required to generate a rule for a subset as the number of Instances In it grows.

Conclusion

The pf &s data base has been explored quite extensively using these techniques. Although It Is the largest data base for which a comprehensive investigation has been conducted, further encouraging results have been obtained when the iterative approach has been applied to other problems. In a recent trial, for example, another set of 20,236 instances each described by 26 attributes was analysed using the first method. A rule was found at the first attempt, using an Initial window size of 400 and an exceptions limit of 100. Twenty Iterations were needed, taking a total of 394 seconds, and the final window contained 2,160 instances (or about 7% of the data base). The rule itself was roughly seven times as complex as the one for pins, forks and skewers shown In the Appendix, If the difficulty of an induction task can be approximated by the product

rule complexity × *number of instances* × *number of attributes*

then this problem was more than two orders of magnitude harder than pf&s. Comparison of the time required to solve each problem suggests that it Increases only linearly wlth the difficulty of the Induction task, and experiments with tasks of intermediate difficulty support this conjecture. This approach does not seem to suffer from a combinatorial explosion (such as that noted In [Hayes-Roth and McDermott 1977]) which would prevent Its application to more difficult problems.

These results, together with those of the previous sections, demonstrate that it is Indeed possible to generate a correct rule to explain a large collection of Instances, even when only a small part of this data base can be held in random access memory. The Iterative technique discussed, coupled with either method of selecting the randomly accessible subset of the data, seems to converge quickly. One main advantage of this Iterative technique Is that It requlres little or no modlflcatlon of whatever inductive algorithm Is being used, but only the way In which it Is used.

In fact the results go further than this. The pf&s induction problem **used as an example** throughout this paper contains about two thousand instances. This **was** small enough to fit In random access memory, so the same inductive algorithm used in the above experiments was applied to It directly. The rule (a correct one, naturally, since it was derived from and explained the entire data base) was obtained in 3.6 seconds. This figure Is marginally greater than the time required in many of the trials using the iterative technique **wlth both the first** and second window-selecting methods, and almost ail of those using intermediate values for the parameters (for which the mean time Is 3.3 seconds with standard deviation 0.2). Thus the techniques discussed In this paper, which **may** become a matter of necessity when dealing with a large data base, may also be desirable for reasons of efficiency when **dealing with a** smaller one.

Acknowledgements

The author Is most grateful to Donald Michie of the University of Edinburgh, both for raislng the problem and for the many helpful suggestions he made throughout the project. He also thanks Martin Schairer of Stanford University for several useful discussions.

Ret erences

Buchanan, B.G. and Mitchell, T.M. [1978) Model-directed learning of production rules. In *Pattern-Directed inference Systems,* (Waterman and Hayes-Roth, **Eds.), Academic Press, N.Y.**

Buchanan, B.G., Mitchell, T., Smith, R. and Johnson, C.R.[1979] Models of learning systems. In *Encyclopedia of Computer Science and Technology,* **(Belzer, Hoizman and Kent, Eds.),** Marcel Oekker, N.Y.

Davls, R.[1977] Interactive transfer of expertise: acquisition of new inference rules. *Proc. 5th Int. Joint Conf, Artificial intelligence,* Cambridge **Mass.**

Olehr, G. and Hunt, E.B.[1968] A comparison of memory allocation algorithms in **a** logical **pattern recognizer.** Tech. Rep,, Department of Psychology, University of Washington, Seattle Wash. A summary of this work also appears In Hunt, E.B., *Artificial intelligence,* Academic Press, N.Y. 1976,

Hayes-Roth, F. and McDermott, J.[1977] Knowledge acquisition **from structural descriptions.** *Proc. 5th int. Joint Conf. Artificial intelligence,* Cambridge **Mass.**

Hunt, E.B., Marln, J. and Stone, P. [1966] *Experiments in induction,* **Academic Press, N.Y.**

Klsickl, E. and Nagel, H.-H. [1976] PASCAL for the DECsystem-10. Mitteilung Nr. 37, lnstltut fuer lnformatik der Universitaet Hamburg.

Lenat, D.[1977] Automated theory formation in mathematics. *Proc. 5th Int. Joint Conf. Artiflciai intelligence,* Cambridge Mass.

Michaiski, R.S.[1978] Pattern recognition as knowledge-guided computer induction. Department of Computer Science, University of Illinois at Urbana-Champaign.

Michie, D. and Bratko, I. [1978] Advice table representations of chess endgame knowledge. *Proc. AISB/GI Conf. Artificial Intelligence.*

Nilsson, Nils [1966] *Learning Machines: Foundations of Trainable Pattern-Classifying Systems.* McGraw-Hilt, N.Y.

Quinian, JR [1976) Predicting the length of solutions to problems. *Proc. 4th Int. Joint Conf. Artificial Intelligence,* Tbilisi U.S.S.R.

Vere, S.A.[1978] Inductive learning of relational productions, in *Pattern-Directed Inference* Systems, (Waterman and Hayes-Roth, Eds.), Academic Press, N.Y.

	initial Window Size 40 Exceptions Limit 20		initial Window Size 400 Exceptions Limit 200	
iteration	Window Size	Number of Exceptions	Window Size	Number of Exceptions
1	40	176	400	27
2	60	256	427	18
3	80	117	446	6
4	100	346	460	
6	120	131		
6	140	43		
7	160	64		
8	180	15		
9	196	1		
10	196	-		
	time: 5.4 seconds		time: 4.1 seconds	

Table 1: Detailed results for two extreme cases.

initial Window Size	Exceptions Limit				
	26	60	76	100	126
60	4.0	4.2	4.2	4.2	4.3
100	4.4	3.7	3.1	3.9	3.6
160	3.9	3.0	3.1	3.4	3.8
200	3.4	3.7	3.0	3.2	3.2
260	3.6	3.4	3.4	3.2	3.3
300	4.0	3.2	3.4	3.6	3.3
360	3.9	3.4	3.4	3.6	3.7

Table 2a: Average time to find correct rule (seconds),

initial Window size	Exceptions Limit				
	26	60	76	100	126
60	8.6	6.3	6.1	5.6	5.5
100	7.1	6.6	4.6	6.1	4.0
160	6.8	4.3	4.3	4.6	4.8
200	4.8	4.8	3.9	4.0	4.0
260	4.6	4.1	4.1	3.9	4.0
300	4.6	3.6	3.8	3.9	3.8
360	4.1	3.6	3.6	3.9	3.9

Table 2b: Average iterations to find correct rule.

initiai Window Size	Exceptions Limit				
	26	60	76	100	126
60	207	248	317	376	390
100	226	268	294	360	368
160	242	269	297	321	343
200	280	310	312	332	326
260	316	352	344	366	361
300	362	381	387	387	378
360	406	417	421	412	418

Table 2c: Average final window size.

Window Size	Average Time **to Find Rule**	**Average iterations** to Find Rule
200	10.1	14.0
260	4.6	**5.6**
276	3.4	4.1
300	3.4	4.1
326	3.3	3.8
360	3.4	3.6
376	3.7	3.9
400	3.6	3.6
600	3.9	3.4
600	**4.5**	3.4

Table 3: Average performance over ten runs using the **second method.**

Appendix: A correct rule for the pins, forks and skewers induction problem.

This rule **is** presented as **a** tree turned on its side. **Each interior node is an attribute name,** and the branches corresponding to the various values of the attribute **appear underneath it.** Each leaf is either **a class** *(lost* or safe) or ***null.***

The attribute names and their meanings are as follows:

distance k-n	distance in king moves from the black king **to the** black knight **('3'** means 'greater **than 2')**
distance k-r	ditto black king to white rook
distance k-wk	ditto black king to white king
distance n-r	ditto black knight to white rook
distance n-wk	ditto black knight to white king
distance r-wk	ditto white rook to white king
r threatens k	white rook checks black king
r threatens n	white rook threatens black knight
k can approach	black king can move adjacent to black knight
k at **p2**	black king is on an edge one square from a corner
wk at **p3**	white king **is** on an edge two squares from **a** corner
wk at **p5**	white king diagonally two squares from **a** corner
n **at p4**	black knight is diagonally one square from **a** corner
wk next to r tine	white king is in **a** rank or file **adjacent to the one** occupied by the white rook

Rule for pins, linear forks and skewers

```
k can approach
       ‘distance n-wk
              distance k-r
               1↦safe
                   k at p2
              2↦        t↦   n at p4
                               t↦safe
                               fw’ost
                        fw’ost
  t↦   1↦          n at p4
              3↦        t↦   k at p2
                               t↦safe
                               f↦lost
                        f↦lost
       2↦safe
       3↦safe
       distance k-r
              distance r-wk
                  wk at p3
       1↦     1↦        t↦   k at p2
                               t↦safe
                               fwlost
                        fwlost
              2↦safe
              3↦safe
              distance r-wk
                 wk at p3
                         k at p2
                                distance n-r
              1↦        t↦     t↦   1↦lost
  f↦   2↦                           2↦null
                                    3↦safe
                               f↦lost
                        f↦lost
              2↦lost
              3↦lost
              wk at p3
                 ‘distance k-wk
                         k at p2
                                wk next to r line
                                      distance n-wk
       3↦     t↦   2↦        t↦     t↦   1↦lost
                                         2↦null
                                         3↦safe
                                      f↦lost
                               f↦lost
                   3↦lost
              f↦lost
```