A Comparative Study of Inductive Learning Systems AQ11P and ID-3 Using a Chess Endgame Test Problem

by

Paul O'Rorke

File No. UIUCDCS-F-82-899

ISG 82-04

A Comparative Study of Inductive Learning Systems AQ11P and ID-3 Using a Chess Endgame Test Problem

by

Paul O'Rorke

September 1982
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ISG 82-2

File No. UIUCDCS-F-82-899

This work was supported in part by National Science Foundation Grants No. MCS 79-06614 and MCS 82-05166.

Acknowledgements : This report grew out of a class project for Prof. Donald Michie's Expert Systems course (Perspectives for Knowledge-Based Systems) CS397DM - Fall 1980 at the University of Illinois at Urbana. Jeff Jackson struggled with the author on the original project, and was always helpful. We are greatful to Tim Niblett for providing us with ID-3, the chess data and an early copy of the Niblett-Shapiro paper. Tim also contributed quite a bit of time helping the author muddle through many difficulties. Robert Stepp provided valuable conversations with the author, as did Prof. Ryszard Michalski. Prof. Michalski was the author's main source of support and encouragement. He also contributed very helpful comments on an earlier draft of this report. Mike Stauffer found errors which led to improvements in the technical results of section 5.1.

Contents

1 Introduction

In many important problem areas, the excellent performance exhibited by human experts seems wholly dependent on their use of large stores of special knowledge. It is sometimes possible to formalize this knowledge as a collection of SITUATION-->ACTION ("if SITUATION then ACTION") rules. Current rule-based expert systems (computerized consultants) are designed to take advantage of these facts. They generally have simple "inference engines" but while they may be "short on brains" in this sense, they make up for it by being "long on knowledge." Their proficiency proceeds from their "knowledge base",(a large collection of fairly simple SITUATION-->ACTION rules).

Much of the work done by knowledge engineers involved in the construction of computerized consultants has been devoted to the extraction and formalization of selected specialist's expertise. Presently, acquisition of knowledge from human experts is so difficult, slow, and costly that many consider it to be "the bottleneck problem in the building of applications-oriented intelligent agents." [FEIGENBAUM 78]

Inductive learning systems, which generate PATTERN-->CLASS decision rules given examples of objects from several classes, promise to widen the knowledge-acquisition bottleneck. Early results indicate that, in some cases, it is not only easier to generate rules by automatic "learning from examples" [NIBLETT and SHAPIRO 80] [QUINLAN 79,80] but that the rules produced by inductive learning programs may be superior to any created (directly) by people [MICHALSKI and CHILAUSKI 80].

Since human beings very often find it easier to learn and to teach classification rules by use of examples, inductive learning procedures may play an important role in the construction of rule-based expert systems. At present, however, "the right way to handle computer induction so as to generate humanly transparent decision structures is an open question" [MICHIE 80B]. Two systems representing quite different approaches (R.S. Michalski's AQ11 and J.R. Quinlan's ID-3) have been successful on a number of non-trivial problems. In this report, we review and compare their methods, and we consider their performance on a chess endgame pattern learning exercise. Our studies lead us to an understanding of the methods and their strengths and weaknesses. This in turn causes us to suggest improvements.

To ensure that the reader has sufficient background to read the rest of this report, the remainder of this section is devoted to basic terminology.

Features are basic properties (like color) with finitely many values (eg. red,blue,yellow). Objects are represented by feature vectors (elements of a finite cartesian product of sets of feature values, known as a "feature space"). Usually features are chosen so that many objects are represented by (collapse into) each feature vector.

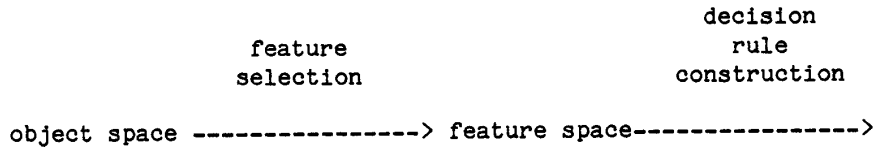
Examples are objects, feature vectors, or sets of either kind, depending on the context. Sometimes the distinction is not important. Classes are (named) sets of examples. In this report, classes are assumed to be disjoint. Pairs of examples with their corresponding class names will be called "observations."

A set of features is adequate if no feature vector stands for objects from more than one class. This means the features are enough to discriminate between the classes.

Decision Rules map examples into classes. Classification is the process of deciding the class of a given example (eg., by "execution" of a decision rule).

The concept of a name is a decision rule determining whether a given object is in the class denoted by the name (following Church (1956) as quoted in [Hunt and Hovland 61]).

The following illustrates the setting for this report:



The methods studied here begin with a given set of features and classified examples. Their task is to compute a more compact representation of a decision function given in the form of classified examples. This has been called pattern recognition, pattern learning , concept learning, description formation (because the decision rules "describe" classes), and inductive learning (because compact decision rules are usually more general than the examples of decisions they were derived from: they assign classes to unobserved examples).

For a look at pattern recognition as rule guided inductive inference see [MICHALSKI 80].

2 The Methodologies

How can a computer be made to "learn" classification rules from observations of classified examples? This section describes two different induction methods (AQ and IP) and implementations (programs AQ11P and ID-3) developed by R. Michalski and J. Quinlan and their collaborators. First the representations used for decision rules are set forth, and then the rule construction algorithms are presented.

2.1 Decision Rule Representations

Like Hunt's Concept Learning Systems, Quinlan's IP (Iterative Polychotomization) methodology uses decision trees to represent decision rules. Each internal node is labelled with a test on some feature. Edges out of a node are labelled with the possible values of the feature. Classes label the leaves. The nodes correspond to sets of objects. The arcs are feature=value associations. A descendent of a node corresponds to the subset (of the set corresponding to the parent node) consisting of objects satisfying the predicate corresponding to the arc from the parent node to the descendent.

Any given object determines (through a sequence of tests indicated by the nodes) a unique path from the root of a tree to a tip. The class label on the tip indicates the class assigned to the object.

In this report, decision trees are usually plotted, but occasionally it is more convenient to use the alternative representation illustrated :

```
RULE                                    ALTERNATIVE EXPRESSION

if feature = value1 then class1         feature
             .                             value1 : class1
             .                                    .
             .                                    .
if feature = valuen then classn             valuen : classn
```

Michalski represents decision rules as lists of descriptions of classes. An object is assumed to be in a class if it satisfies the description of that class. The descriptions are not arbitrary, they must be disjunctions of conjunctions of basic predicates. Michalski calls the descriptions "covers." A cover is a disjunction of "complexes." A complex is a conjunction of "selectors." A selector is an elementary statement asserting that the value of a feature is in some subset (called the "reference" of the selector) of the feature's "domain" (set of possible values).

A cover of A against B is a discriminant description: it is true of all elements of A and false for all elements of B that are not also in A. If we have to discriminate between n classes, we can use n-1 covers to decide the class of any given example.

In this report, disjunctions are presented as lists and conjunction is expressed implicitly through juxtaposition or explicitly by means of the symbol &.

a selector	a complex	a cover
[feature1>value1]	[f1=val0] [f2>val1]	1. [f1>0] 2. [f1=1] [f2<3]

The next sections examine some connections and contrasts between decision trees and covers.

2.1.1 Connections

Here similarities between covers and decision trees are explored.

1) Any decision tree can be transformed into covers whose terms (complexes) are paths in the tree and which yield the same decisions as the tree.

For example, the top level decision tree from [NIBLETT and SHAPIRO 80]:

```
canrun
   f: mainpatt
         f: rookpawn
               f: rank56
                     f: rank7
                           f: interfere
                                 f: DRAWN
                                 t: LOST
                           t: LOST
                     t: LOST
               t: LOST
         t: LOST
   t: LOST
```

is equivalent in this sense to the two covers

DRAWN:

[canrun=f][mainpatt=f][rookpawn=f][rank56=f][rank7=f][interfere=f]

LOST:

[canrun=t]
[canrun=f][mainpatt=t]
[canrun=f][mainpatt=f][rookpawn=t]
[canrun=f][mainpatt=f][rookpawn=f][rank56=t]
[canrun=f][mainpatt=f][rookpawn=f][rank56=f][rank7=t]
[canrun=f][mainpatt=f][rookpawn=f][rank56=f][rank7=f][interfere=t]

2) On the other hand, a set of covers can not necessarily be transformed into a tree whose paths are the terms of the covers.

For example:

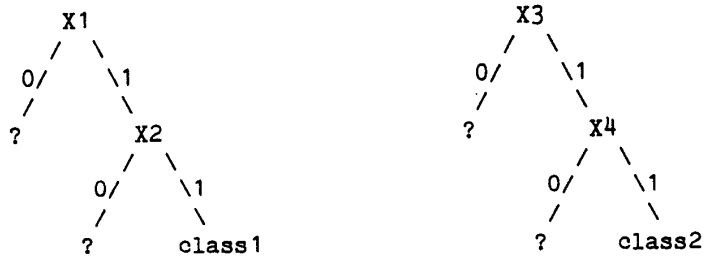
[x1=1][x2=1]--->class1

[x3=1][x4=1]--->class2

cannot be mapped into a single tree whose paths are the terms in the cover since the root node must split on just one feature and there is no feature common to all the terms in the covers.

3) On the third hand, if we allow indecision leaves, sets of covers can be converted naturally to forests of trees whose paths include the covers' terms.

For the covers just above:



4) It is also possible to construct single decision trees for covers, but the correspondence is not as natural as for forests. [MICHALSKI 78] contains an algorithm which constructs "quasi-optimal" decision trees from covers.

5) Trees and covers can be profitably viewed as discrimination nets. Later we will define a unifying decision structure in order to compare trees and covers in a common setting.

2.1.2 Contrasts

Having established connections between covers and decision trees by examining their similarities, we now point out differences.

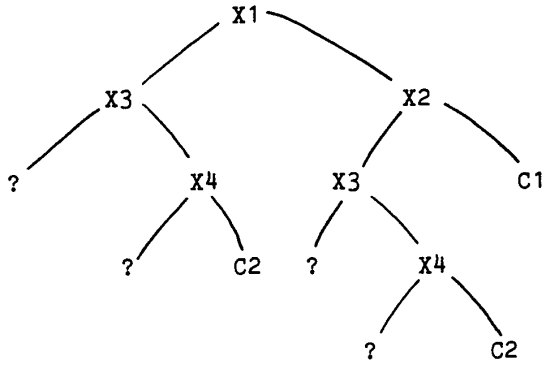
1) Decision trees are structurally more restricted than covers: at any node ID-3 must split on just one feature , and on all values of that feature. This makes it more difficult to represent disjunctions of dissimilar conjuctive concepts.

For example: when X1,...,X4 are binary features,

[X1=1][X2=1]-->c1

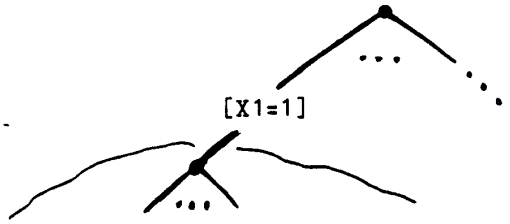
[X3=1][X4=1]-->c2

must be represented as something like



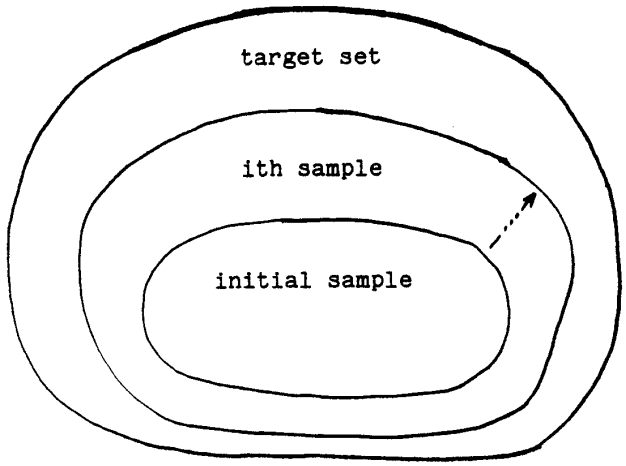
The left branches correspond to feature value 0, and the right branches are implicitly labelled [Xi=1].

2) On the other hand, the tree structure has the advantage that propositions need only appear once to be conjoined with all paths emanating from them (whereas, in a cover, they would have to appear in <u>every</u> term corresponding to a path):



2.2 Decision Rule Construction

AQ and IP both have basic algorithms for constructing a decision rule from a fixed set of examples. In addition to this "core" process, both are capable of some form of incremental generation of decision rules. In an incremental scheme a decision rule for a given set of examples (the "target set") is determined "a bit at a time": first, a decision rule for a subset of the target set (the "initial sample") is built by the core; then the current sample is expanded and a new rule is generated for it, etc. After a finite number of expansions, a rule is produced which correctly classifies every example in the target set.

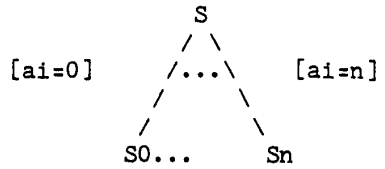


Incremental Rule Generation
ith decision rule is correct for all examples in the ith sample

2.2.1 IP

The core of IP constructs a decision tree for a given sample by associating it with a root and then recursively sprouting nodes associated with multi-class sets (sets of examples from more than one class) attempting to split the sets into uni-class subsets. A set is split into subsets by testing on a feature (not yet tested) which minimizes the average uncertainty about the class of an arbitrary example (from the split set). That is, the feature which yields the most information about the class is observed. By "uncertainty" and "information" we mean entropy and mutual information as defined in information theory [McEliece 77]. By "examples" we mean feature vectors in this case.

Note that IP splits on exactly one feature at a time and furthermore, on all values of that feature. The decision trees it produces must have an arc for each feature=value association on just one feature emanating from each nonterminal node:



No feature will appear twice on any path from the root to a tip of the tree.

IP is "Iterative" because it was designed for induction on large sets of examples. Instead of attempting to construct a decision tree for large sets all at once, a managable initial subset called a "working-set" or "window" is selected randomly. IP iteratively finds a decision tree for the window, tries out the tree out on the entire target, incorporates some exceptions (incorrectly classified examples) into the window, then finds a new rule for the expanded window, etc. Usually this incremental rule generation process converges on a decision rule for the target before the window grows to be too large.

Note that IP is top down: it describes classes by breaking down the description of the whole feature space into "conjunctive concepts" corresponding to the paths of the decision tree. It barely uses individual examples, relying instead on relative frequencies of the classes for guidance in decision rule construction.

2.2.2 AQ

In sharp contrast, the AQ methodology builds decision rules by operating on the descriptions of individual examples. An AQ example is a complex, so it may represent many, not just one individual feature vector. An AQ decision rule for n classes consists of covers of n-1 classes. Algorithm Aq generates a cover of a class (set of positive examples) against all other classes (negative examples) by selecting good complexes which cover positive examples from an approximate description of the complement of the set of negative examples. To be more specific, we give the following annotated Lisp outline of the Aq algorithm. The capitalized Lisp keywords are explained in [Charniak, Riesbeck, and McDermott 80].

Algorithm Aq.
(constructs a decision rule for a class)

Data Structures : l--- lists of complexes
c--- complexes (examples)
s--- selectors

To cover a list of positive examples against a list of negative examples, find the best complex in a cover of the first positive example and add this to a cover of the positive examples not yet covered.

```
(DE cover:l/l (l+ l-)
    (COND ((NULL l+))
          (T (LET (best (bestcomplex (cover:c/l (CAR l+) l-)))
                (CONS best (cover:l/l (knockout l+ best) l-]
```

To cover a positive example against a set of negative examples cover it against the first AND cover it against the rest. "Multiply" computes the conjunction of two covers, returning a cover which is then simplified by absorption and trimming (deletion of excess conjunctions of low value).

```
(DE cover:c/l (c+ l-)
    (COND ((NULL l-) T)
          (T (simplify (multiply (cover:c/c c+ (CAR l-))
                                 (cover:c/l c+ (CDR l-]
```

To cover a positive example against a negative example look along the dimensions of the feature space and return a list of generalizations of the positive example's selectors which don't overlap with the corresponding selectors of the negative example.

```
(DE cover:c/c (c+ c-)
    (FOR (p IN projections)
         (FILTER (cover:s/s (p c+) (p c-]
```

Note that if one tries to cover a positive complex against a negative complex when all their selectors overlap, then cover:c/c will return NIL (FALSE). This reflects the fact that it is impossible to generalize an example against another which intersects it. To avoid this situation, the classes must be disjoint: no negative example should overlap any positive example. If two sets of examples overlap, consider them as sets of feature vectors and define

cover (eplus eminus) := cover (eplus eminus-eplus).

That is, if eplus and eminus overlap, then subtract the feature vectors covered by positive examples before computing the cover of eplus against eminus. This is approximated in implementations by deleting any negative example which intersects with some positive example before computing cover:1/1.

Incremental rule construction techniques also exist based on algorithm Aq. In particular, a method of forming covers using initial guesses has been developed and implemented in program AQ11P. These "initial hypotheses" might be gifts from humanity (derived through special knowledge or intuition), or the results of a previous run on a subset of the examples. New covers are obtained from old (possibly erroneous) covers:

AQ Incremental Rule Generation.
(rule-refinement)

For each class i of m classes,

INPUT:

E_i : a list of examples (complexes)

H_i : a hypothesis (cover)

OUTPUT:

H'_i : a new hypothesis, which correctly classifies the input examples

PROCEDURE: Consider the inputs as sets of feature vectors.

1) Determine inconsistencies. Compute the examples which should be, but aren't covered. They must be "added".

$$E^A_j = E_i - H_i$$

Compute the examples which shouldn't be, but are covered. They must be "subtracted".

$$E^S_{ij} = E_j \cap H_i \qquad \text{for all } j \neq i$$

2) Determine "subtract" covers.

$$H^S_{ij} = \text{Cover}\left(E^S_{ij} , \bigcup_{k=1}^{n} H_k \cup E^A_k \right)$$

$$H^S_i = \bigcup_{j \neq i} H^S_{ij}$$

3) Determine output covers.

$$H'_i = \text{Cover}\left(E_i , \bigcup_{k \neq i} [(H_k - H^S_k) \cup E_k] \right)$$

If rule-refinement is a process which takes a decision rule and a list of classified examples as input, and which uses both to produce a new decision rule consistent with the observation list, then the AQ incremental cover generation method can be called rule-refinement. However, it is misleading to say that the AQ method modifies the initial decision rule to make it consistent with the observations. The new decision rule is not constructed by making corrective changes to the old rule. No attempt is made to retain parts of the initial decision rule consistent with new observations. It is more accurate to say that the AQ method uses covers to make the construction of new covers more efficient. The method takes advantage of the fact that covering a positive example against a complex which represents many negative examples is easier than covering it against a list of those examples.

2.3 Credits

Michalski originally developed a slightly different version of the Aq algorithm presented here in the context of switching theory. The original version provided a lower bound on the number of complexes in the smallest possible cover, so the algorithm gave an upper bound on the difference of the numbers of complexes of the approximate and optimal solutions. The version presented here is only slightly simpler, but it requires less computation and storage. [MICHALSKI 69,71,75A] [MICHALSKI and McCORMICK 71]

James Larson constructed the implementation AQ11 in PL1, including "incremental generation of hypotheses." [LARSON 77] [MICHALSKI and LARSON 78] AQ11P is a translation of Larson's program into Pascal by Brounell and Soukup. [BROUNELL and SOUKUP 80?]

Accounts of applications of AQ include [MICHALSKI and CHILAUSKI 80] (plant pathology) and [MICHALSKI and NEGRI 77] (KPK chess endgame).

Quinlan originally developed and implemented the IP method. It is based on Hunt's Concept Learning Systems [Hunt et al. 66] and thus its roots go back to [HOVLAND and HUNT 61] and the psychological studies of [BRUNER et al. 56].

Alen Shapiro wrote the implementation ID-3 (Iterative Dichotomizer-3) used in the experiments of the next section. The Dichotomizer is so-named because it constructs decision trees which descriminate between exactly two classes, (but features may have more than two values).

IP has been applied to a number of chess endgame pattern learning problems. [QUINLAN 79,80] [NIBLETT and SHAPIRO 80]

3 Experimental Comparison

The AQ and IP methods have different origins, motivations, and scopes, but the fact that they both work in essentially the same setting makes experimental comparison of their performances on a common problem possible. The results of comparison experiments could be misleading, because the implementations could be inefficient or unfaithful representations of the methods, and because the test problem might favor one method over another. For example, the features for our benchmark problem were developed with ID-3 in mind. The fact that they are all predicates (two valued), may make the problem biased against AQ since much of the power of selectors is lost. In spite of the danger of over-generalization, the experiments performed in this section do provide useful information.

All experiments were done on a CDC CYBER 175 under NOS 1.4. All Pascal programs were compiled under release 3 of the Pascal 6000 compiler. Lisp support routines were written in UT Lisp 1.4. ID-3 takes lists of feature vectors with associated classes (encoded in mixed radix notation) as input. AQ11P takes two lists (DRAWN & LOST) of (unencoded) feature vectors occasionally with covers from previous runs. Since all features in the problem described below are predicates, the selectors are binary [feature=value] associations. They are treated as propositional logic literals.

3.1 The KPK-BTM problem.

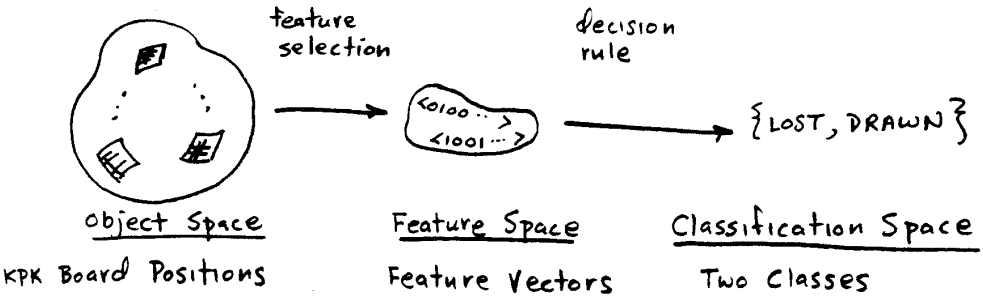
Board positions in the chess endgame in which White has his king and a pawn but Black has only his king (Black to move), can be classified as either LOST for Black or DRAWN. The KPK-BTM configuration below, for example, is DRAWN because Black can capture White's pawn immediately.

```
. . . . . .
. . . . . . B
. . . . . . P
. . . . . .
. . . . . .
W . . . . . .
. . . . . .
. . . . . .
```

At the Machine Intelligence Research Unit of the University of Edinburgh, Tim Niblett and Alen Shapiro developed a set of thirty-one propositional (true or false) features adequate for discriminating the

LOST and DRAWN positions. One such feature is IMCAP, which has value 1 (true) if Black can capture the pawn immediately (as in the example above) and 0 (false) if he cannot. For a list of their features, see Appendix A. Seen through these features, the 83,622 legal KPK-BTM positions collapse into 2411 feature vectors. Of these, 1045 vectors represent DRAWN sets of board configurations and 1366 represent LOST positions.

FIG. 3.1 - THE KPK-BTM CHESS ENDGAME PATTERN LEARNING PROBLEM



The KPK-BTM chess endgame pattern learning problem is:

GIVEN the complete set of feature vectors representing the lost and drawn board positions,

FIND a simple rule for deciding when board positions are lost or drawn.

A solution to the problem might be a rule of the form :

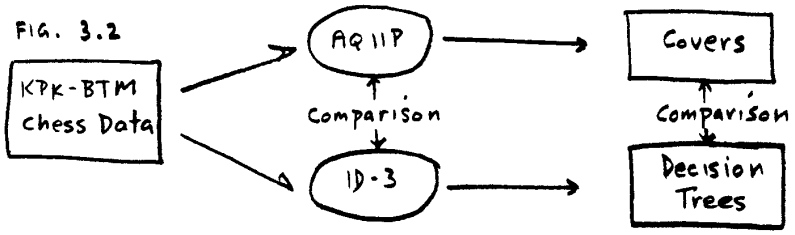
if IMCAP or ... , then DRAWN ;
else LOST.

The next section tells how the KPK problem just described was used as a benchmark to compare two inductive rule generation systems.

3.2 The Comparison Experiments

The two inductive rule generation systems were used to manufacture rules for correct classification of every feature vector in the KPK data. Measurements of the cost of producing decision rules by several methods were recorded. Then the rules were evaluated to yield some qualitative and quantitative aspects of their quality.

FIG. 3.2



To avoid confusing differences in the basic description-forming "cores" of ID-3 and AQ11P with differences in their incremental rule generation strategies, the two programs were first compared without incremental rule generation (experiment one), and then the different incremental strategies were compared (in experiment two).

In the third experiment rule-refinement was tried and the comprehensibility of the simplest decision rules induced by each program was considered.

3.3 Experiment One: Base Comparison

The purpose of this experiment is to compare the induction mechanisms which form the cores of AQ11P and ID-3. The experiment requires that both programs construct decision rules for correct classification of all given feature vectors without the benefit of incremental rule generation.

Method 1: AQ11P was run on the entire set of examples with no initial hypotheses. ID-3 was also run on the full data set (with the initial window size set to 2411).

Results 1: AQ11P produced the decision rules of figure 3.3. ID-3 produced the decision tree shown in figure 3.4. The list of node labels in figure 3.4 is in prefix order.

Figure 3.3. AQ11P Decision Rules.

```
* * * Cover of Class 1 (DRAWN) * * *

 1.  cimmt
 2. ~sint- &~spra7 & mnear & rnear &~rneac &~cplu1
 3. ~mmp1- & cpat2
 4.  srfil & rstal
 5.  srfil & rrp2-
 6.  srfil & rneac & cwksa
 7. ~sint- &~5r6p- & 5btop &~5diro &~mpmov &~mmp2- &~mmp1- & cahea
 8.  srfil &~mpmov &~mnear & rpsq- & rneap
 9. ~sint- &~spran &~spra7 &~mpmov &~mmp2- &~mmp1- &~mdiro & cplu2
10. ~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~ccrit &~cahea & cplu2
11.  srfil &~spra7 &~5r6p- &~many- & rpsq- & rneac
12.  5btop & many- & rpsq- &~rneap
13.  srfil &~spran &~mnear & rpsq- & rnea8
14. ~spran &~5btop & many- &~ccrit & cplu2
15. ~5r6p- & 5btop &~mmp1- &~mdirb & mnear & cplu1
16.  srfil & 5r6p- & cplu1
17. ~5mp-- & 5btop & mdirb & ccrit

* * * Cover of Class 2  (LOST) * * *

 1. ~cpat2 &~cwksa &~cahea &~cplu2 &~cimmt
 2. ~srfil & mmp1- &~mnear
 3.  mpmov &~rstal &~rrp2- &~rneac &~cpat2 &~cimmt
 4.  5mp-- &~mpmov &~rstal &~rnear &~rneac &~cahea &~cimmt
 5. ~srfil & spran &~5btop &~cpat2
 6.  spra7 &~5btop &~cpat2
 7. ~srfil & 5r6p-
 8.  5diro & rneap
 9. ~srfil & mmp2- &~cpat2 &~cimmt
10. ~srfil & mdiro
11.  sint- &~srfil &~many-
12.  srfil &~rnear &~rneap & ccrit &~cwksa &~cplu1
13. ~srfil & mpmov &~mmp2- &~cimmt
14.  5mp-- &~mpmov &~mnear &~rnear &~rneap &~rnea8 &~rneac & cahea
15.  sint- & rrp1-
16. ~srfil &~mmp2- & mmp1-
17. ~srfil & 5mp-- & 5btop & mdirb &~cahea
```

Figure 3.4. ID-3 Decision Tree.

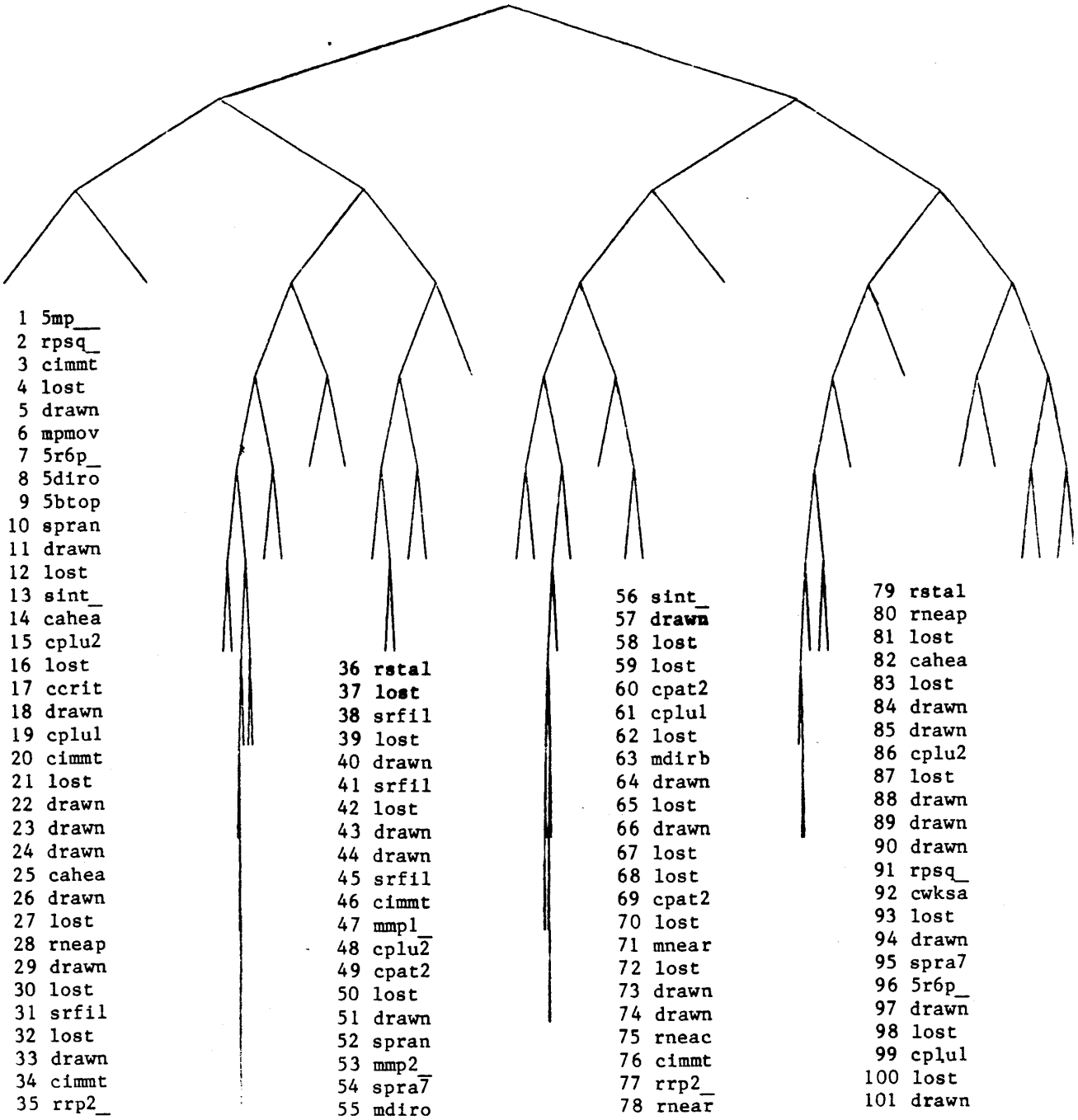


Figure 3.5. Experiment 1 Comparison Table.

	ID-3 DECISION TREE	AQ11P COVERS
no. of conjunctive subconcepts	(paths)	(complexes)
drawn	26	17
lost	25	17
total vs. avg.	51	17
avg. conjunction size		
drawn	6.9	4.6
lost	7.2	4.1
total	7.1	4.3
max. conjunction size		
drawn	12	8
lost	12	8
total	12	8
min. conjunction size		
drawn	4	1
lost	4	2
total	4	1
number of different features tested	27 (all but 12,19,22,25)	29 (all but 19,25 or 20,25)
% correctly classified	100%	100%
rule generation time (CPU SECONDS)	5	325
rule generation storage (DECIMAL NO. OF 60 BIT BINARY WORDS)	23036	37824

Consider the paths of the decision tree and the cómplexes of the covers as "conjunctive subconcepts" of the decision-rule "concepts". The number of arcs in a path and the number of selectors in a complex naturally measure the "size" of the corresponding conjunctive subconcepts.

Figure 3.5 shows that there are significantly fewer conjunctive subconcepts in the rules produced by AQ11P, and that the subconcepts are significantly smaller. The tree produced by ID-3 tests slightly fewer features than either of the rules produced by AQ11P. Since both programs were allowed to observe the entire set of feature vectors representing all possible legal KPK board positions, the resulting rules correctly classify 100% of those feature vectors. Finally, AQ11P used about 1.6 times as much storage space and 65 times the time required by ID-3 to generate a decision rule.

3.4 Experiment Two: Sequential Learning Comparison

Having seen the operation of the central components of AQ11P and ID-3 on all the examples, we move on to compare their incremental rule generation capabilities.

Method 2: A subset (which we will call "SAMPLE") of 500 of the 2411 examples was chosen randomly from the KPK-BTM data by a LISP selection program written by the author. ID-3 and AQ11P were both run on SAMPLE, producing the results in figures 3.6 and 3.7. Next they were both allowed to add up to 200 exceptions to SAMPLE, and a second pass was made to produce decision rules for the expanded set of examples. An extra run of AQ11P was made using the covers generated in the first pass as initial hypotheses. See figures 3.8,3.9,and 3.10. Finally, complete decision rules were generated iteratively: ID-3 was run using Quinlan's expansion strategy with an initial sample of 500 examples (this time chosen randomly by ID-3) with as many as 500 feature vectors added in each iteration, (fig 3.11).

The two previous runs of AQ11P in this section were designed to find out how it might behave if it had a built-in incremental rule generation strategy like Quinlan's. The fact that ID-3 required six iterations for convergence pursuaded the author to discontinue simulating Quinlan's strategy using AQ11P. Instead, a complete decision rule was generated by expanding the initial SAMPLE to the entire data base on the second pass, using the decision rules constructed in the first pass as input hypotheses (fig. 3.12). For convenience, this method will be called the "single expansion" strategy.

Figure 3.6. ID-3 Decision Tree from Initial Pass on SAMPLE.

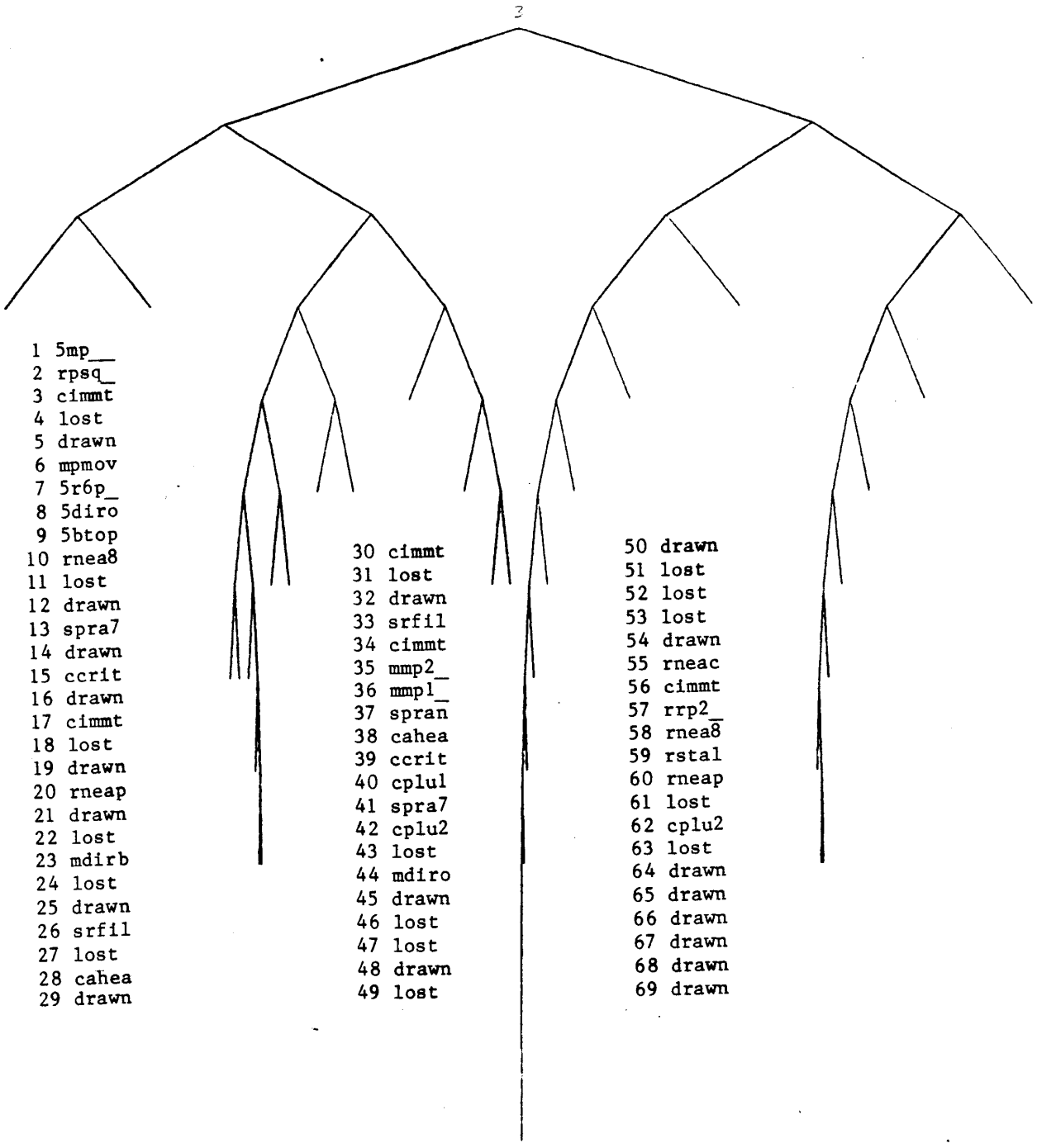


Figure 3.7. AQ11P Initial Pass on SAMPLE.

```
                * * * Cover of Class 1 (DRAWN) * * *

 1.  cimmt
 2.  srfil & spran &~mmp1- & rpsq-
 3.  5btop &~rnear & rrp2-
 4. ~spran &~spra7 &~mpmov &~mmp2- &~mmp1- &~mdiro & rpsq-
 5. ~srfil &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov & rpsq-
 6.  srfil & rpsq- & rneac &~ccrit
 7.  srfil &~spra7 & ccrit &~cahea & cplu2
 8.  spra7 &~mdirb & cplu1

                * * * Cover of Class 2  (LOST) * * *

 1. ~srfil & mmp1-
 2. ~rstal &~rrp2- &~cahea &~cplu2 &~cimmt
 3. ~srfil & spran &~5btop
 4. ~sint- & 5mp-- &~mmp1- &~many- &~mnear &~rstal &~rnear &~rrp2- &
    ~rneap
 5. ~srfil & rneap &~rneac &~cahea &~cimmt
 6. ~spran &~mmp2- &~rstal &~rnear &~rrp2- &~rneap &~rnea8 &~cplu1
 7.  5r6p- &~mdirb
 8.  mdiro &~cplu1
 9. ~srfil & 5diro
10. ~spran &~mmp1- &~rstal &~rnear &~rrp2- & ccrit &~cahea &~cimmt
11. ~srfil & mmp2- &~cimmt
12.  mpmov &~rstal &~rrp2- &~rneac &~cimmt
13. ~5btop &~rnear &~rneac
14.  spran & 5mp-- & mnear &~ccrit
```

Figure 3.8. ID-3 Second Pass on SAMPLE & Exceptions.

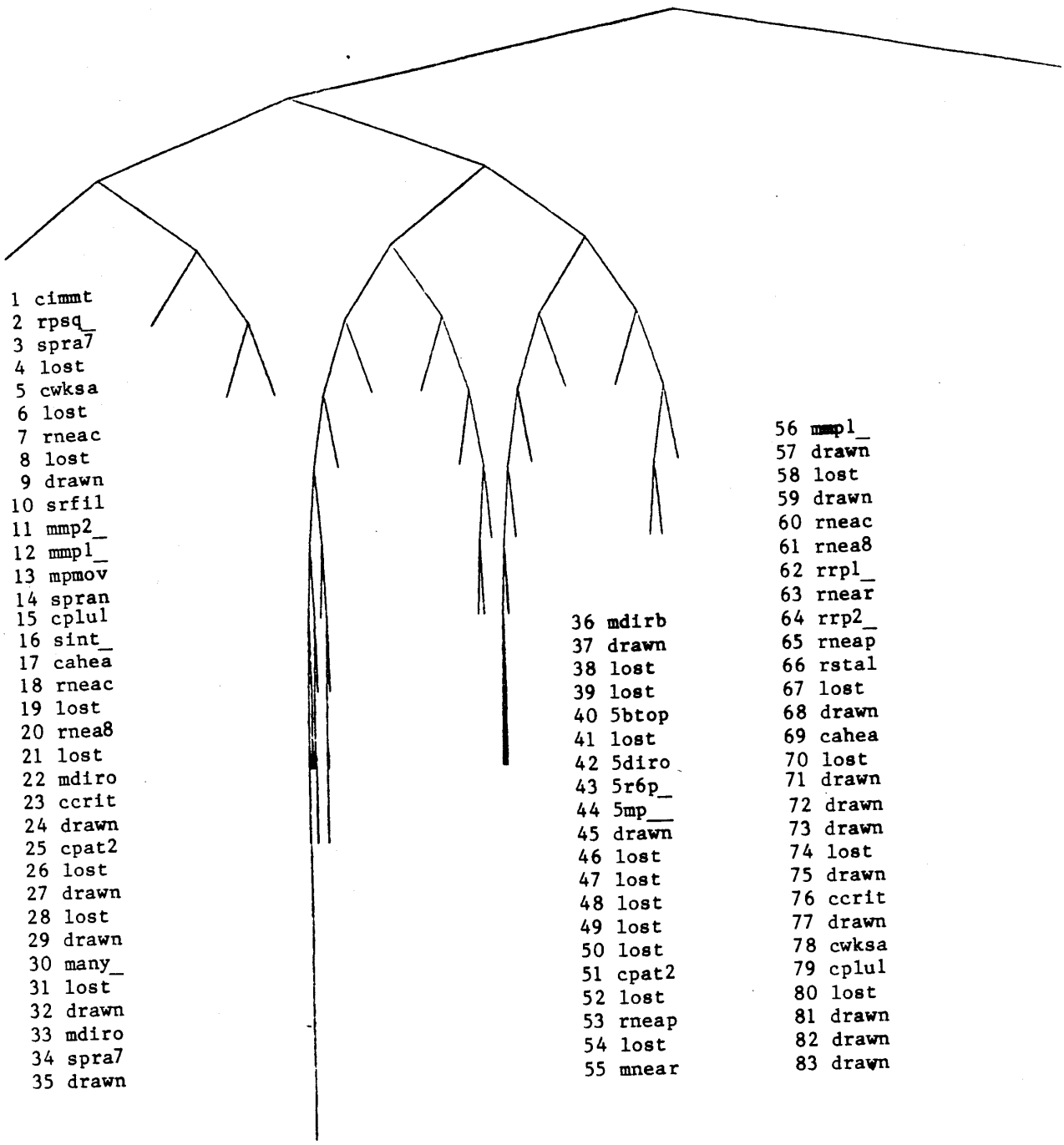


Figure 3.9. AQ11P Second Pass on SAMPLE & Exceptions (without rule-refinement)

* * * Cover of Class 1 (DRAWN) * * *

1. srfil &~spra7 &~5r6p- &~many- & rneac
2. 5btop &~mpmov & rneap & ccrit & cahea
3. ~sint- &~5r6p- & 5btop &~5diro &~mpmov &~mmp2- & rnear & rpsq-
4. ~spran &~5mp-- &~mpmov & cplu1
5. srfil & rpsq- & rnea8 &~ccrit
6. ~spran &~mmp2- & rnear & rneac & cplu2
7. ~spran & 5mp-- & 5btop &~mpmov & cwksa &~cahea
8. srfil & rstal
9. ~mmp1- &~mnear & rpsq- & rneap &~ccrit &~cplu2
10. spran & mnear & rpsq- &~cplu2
11. cimmt
12. srfil & rrp2-
13. ~sint- &~5mp-- & mnear & cahea &~cplu1
14. ~sint- &~spran &~mmp2- &~mmp1- &~mdiro & rnea8 & rneac &~ccrit & cplu2
15. spran &~5r6p- &~5mp-- & 5btop &~5diro & cplu2
16. sint- & many-
17. ~mpmov & rnear &~rneap &~rneac

* * * Cover of Class 2 (LOST) * * *

1. ~srfil & mmp1- &~cpat2
2. ~mmp1- &~rstal &~rrp2- &~cpat2 &~cahea &~cplu2 &~cimmt
3. ~spran & mpmov &~rstal &~rrp2- &~rneac &~cimmt
4. spran &~5btop &~mdiro &~mdirb &~cpat2 &~cwksa
5. 5mp-- &~rstal &~rrp2- &~rneap &~rnea8 &~rneac &~cimmt
6. ~srfil & 5mp-- &~rnea8 &~ccrit &~cimmt
7. ~srfil &~spran &~rneac &~ccrit &~cplu1 & cplu2 &~cimmt
8. 5r6p- &~mdirb
9. mdiro &~cplu1
10. ~srfil & spran & 5mp-- & mdirb
11. ~srfil & 5diro
12. ~srfil & 5mp-- &~cpat2 & ccrit &~cahea &~cimmt
13. ~rneap & rneac & ccrit &~cplu1 & cplu2
14. ~srfil & mpmov & cplu1
15. ~srfil & mmp2- &~rneap
16. sint- & mnear
17. sint- &~srfil &~many-

Figure 3.10. AQ11P Second Pass on SAMPLE & Exceptions
(with rule-refinement)

* * * Cover of Class 1 (DRAWN) * * *

```
1.  srfil &~5r6p- & mmp1- &~mnear & rneac & cplu1 & cplu2
2. ~5r6p- &~mpmov &~mdirb &~mnear &~rnear &~rneap & rneac & ccrit
3. ~sint- & srfil &~spran &~5r6p- & 5btop &~mpmov &~mdiro & rneap &
   cahea
4. ~spran &~5r6p- &~5diro &~mmp2- &~mmp1- &~mdiro & rnear & rneac &
   cplu2
5. ~rneap & rneac & ccrit &~cwksa &~cahea & cplu1 & cplu2
6.  srfil &~spran &~5r6p- &~mpmov &~mdiro & rnear &~rrp1- & cplu2
7.  srfil &~5r6p- &~5mp-- & 5btop &~mpmov &~mdiro & rnea8 &~ccrit &
   cplu2
8.  srfil &~spran &~5r6p- &~mdiro & rrp2- & rneac
9.  srfil &~5r6p- & 5btop &~mdiro &~mnear & rstal
10. ~sint- & 5mp-- &~mdirb &~rrp2- & rneap & rnea8 & cpat2 &~ccrit &
   ~cwksa &~cahea &~cplu2
11. ~5r6p- & 5btop &~5diro &~mmp1- &~mdiro &~mnear & rneap & cimmt
12. ~spran & 5mp-- & 5btop &~rpsq- &~rneap &~cahea &~cplu2 & cimmt
13. ~5r6p- &~5mp-- & 5btop &~5diro &~mmp1- &~mdiro & rneap & cimmt
14.  srfil &~5r6p- & 5btop &~mdiro &~mnear & rrp2-
15. ~sint- &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp2- &~mmp1- &
   ~mdiro & rneap & cahea
16. ~sint- &~spran &~5r6p- &~5mp-- &~5diro &~mmp2- &~mmp1- &~mdiro &
    rnea8 & rneac & cahea
17.  spran &~5r6p- &~5mp-- & 5btop &~5diro &~mmp2- &~mmp1- &~mdiro &
    rneac & cplu2
18. ~spran &~5r6p- &~5mp-- &~5diro &~mmp2- &~mmp1- &~mdiro & rnea8 &
    rneac &~ccrit &~cahea & cplu2
19.  srfil &~5r6p- & mmp1- &~mdiro &~mnear & rnea8 & rneac & cplu2
20. ~spran &~5r6p- &~5diro &~mmp2- &~mmp1- & many- &~mdiro & rnea8 &
    rneac &~ccrit & cplu2
21.  sint- & srfil &~5r6p- &~mdiro &~mnear & rnea8 & rneac &~ccrit &
    cplu2
22. ~sint- &~spran &~5r6p- &~5diro &~mmp2- &~mmp1- &~mdiro & rneap &
    rneac &~ccrit & cplu2
23.  srfil &~5r6p- &~mdiro &~mnear & rneap & rneac &~ccrit & cplu2
24.  srfil &~5r6p- & mmp2- & mmp1- &~mdiro &~mnear & rneac & cplu2
25.  srfil & spran &~5r6p- &~5mp-- & 5btop &~mpmov &~mdiro & cahea
26. ~5r6p- & 5btop &~5diro &~mpmov &~mmp2- &~mmp1- &~mdiro & rneap &
    ccrit & cahea
27. ~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp2- &~mmp1- &~rneap &
   ~ccrit & cplu1 & cplu2
28. ~spran &~5r6p- &~5diro &~mmp2- &~mmp1- & mnear & rneac &~ccrit &
    cplu1 & cplu2
29.  srfil &~spran &~5r6p- & rnear & rneac & cplu1 & cplu2
30. ~spran &~5r6p- &~5diro &~mpmov &~mmp2- &~mmp1- & rnear & cahea &
    cplu1
31.  srfil &~5mp-- & mdirb & rneac & cahea & cplu1
32.  srfil &~spran &~5r6p- & rneap & rneac & cahea & cplu1
```

* * * Cover of Class 2 (LOST) * * *

1. ~srfil &~spra7 &~5btop & mmp1- &~cpat2 &~cimmt
2. ~spra7 &~rrp2- &~rpsq- &~cplu2 &~cimmt
3. ~srfil &~rrp2- &~rpsq- &~cplu1 &~cimmt
4. ~rrp2- &~rpsq- &~cwksa &~cplu1 &~cplu2 &~cimmt
5. ~srfil &~spran &~spra7 & mpmov &~rrp2- &~cimmt
6. ~spra7 &~rstal &~rnear &~rrp2- &~rneac &~cahea &~cplu1 &~cimmt
7. ~srfil &~spra7 & 5mp-- & mmp2- &~rrp2- &~cpat2 &~cimmt
8. ~srfil & spra7 &~5btop &~cpat2 &~cplu1 &~cimmt
9. ~srfil & spran &~spra7 &~5btop &~cpat2 &~cimmt
10. ~srfil & spran &~spra7 & 5r6p- &~rrp2- &~cimmt
11. ~srfil &~5btop & mdiro &~cplu1 &~cimmt
12. ~srfil & spran & 5diro & mdirb &~rrp2- &~cimmt
13. srfil &~spran & spra7 &~rrp2- & ccrit &~cplu1 & cplu2 &~cimmt
14. ~spran &~spra7 & 5mp-- & mmp2- &~rrp2- &~rneap &~rneac & cahea & ~cimmt
15. ~srfil &~spra7 & 5mp-- & mmp1- &~rrp2- &~cpat2 &~cimmt
16. ~rnear &~rrp2- &~rneap &~rnea8 & ccrit &~cwksa
17. srfil &~spra7 & mmp1- &~rrp2- &~rneac & cahea &~cimmt
18. ~srfil & spran & 5mp-- & mdirb &~rrp2- &~cimmt
19. ~srfil & spra7 & 5mp-- & mdirb &~rrp2- & cplu1 &~cimmt
20. ~srfil &~rneac &~ccrit & cwksa &~cahea &~cimmt
21. ~rrp2- &~rneac & cwksa &~cahea & cplu1
22. sint- &~srfil &~spran &~many- &~rrp2- & rneap &~ccrit &~cwksa & ~cplu1
23. ~mmp2- & rrp1- &~rnea8 &~rneac &~ccrit &~cplu1 &~cimmt
24. ~mmp1- &~mnear &~rstal &~rnear &~rrp2- &~rneap & ccrit &~cahea

Figure 3.11. ID-3 Decision Tree Correct for All KPK Positions.
(generated iteratively)

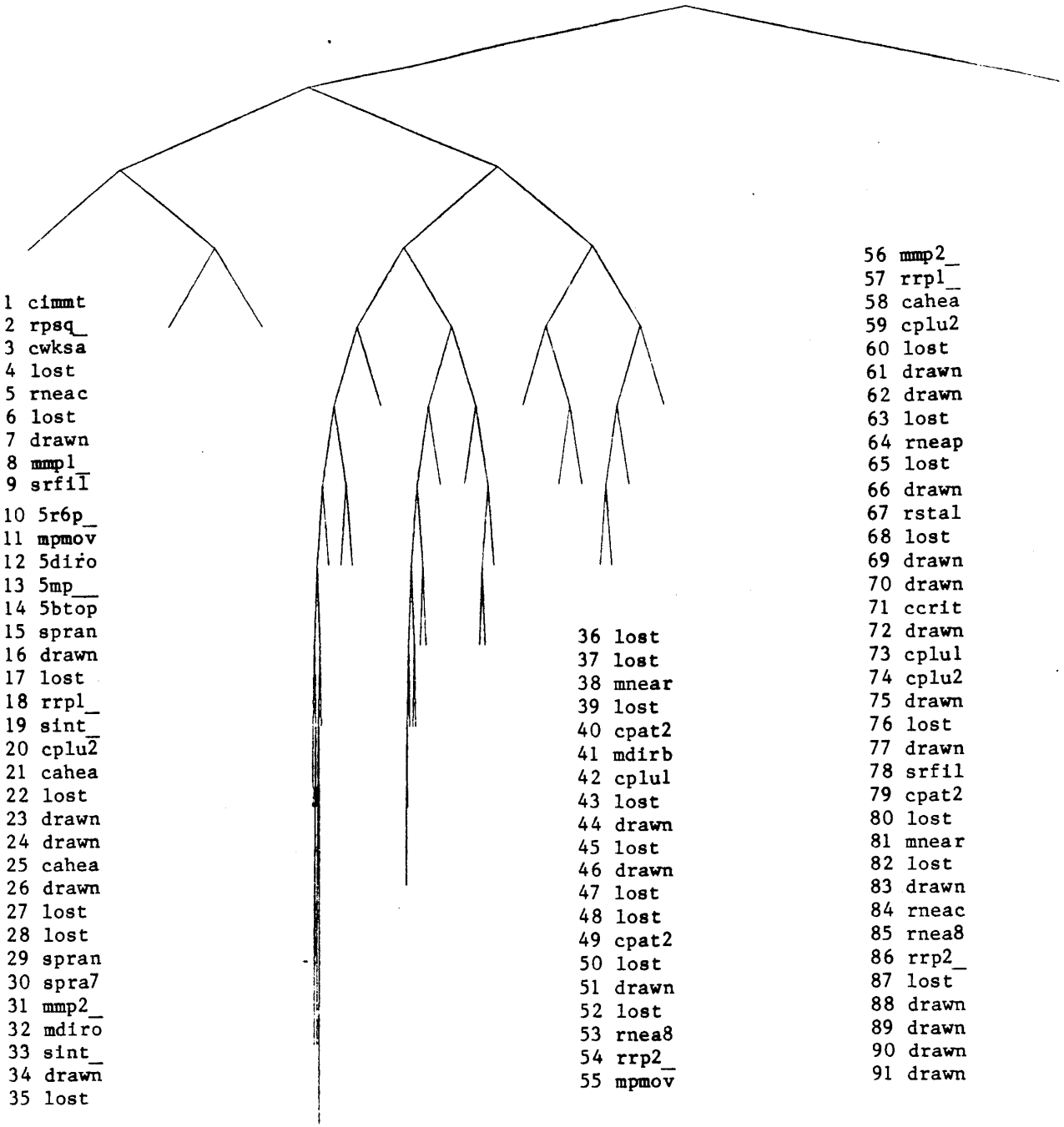


Figure 3.12. AQ11P Single Expansion Covers.

* * * Cover of Class 1 (DRAWN) * * *

```
 1.  cimmt
 2. ~spra7 &~mpmov &~mdirb & mnear &~rrp2- &~rneac & cpat2 &~cpat1 &
    ~ccrit &~cwksa &~cahea &~cplu1 &~cplu2
 3. ~spra7 &~mmp1- &~mdirb &~rrp2- &~rrp1- & rneap &~rneac & cpat2 &
    ~ccrit &~cwksa &~cahea &~cplu1 &~cplu2
 4.  5btop &~5diro & mmp2- &~mmp1- & rstal &~rnear &~rrp2- &~cahea &
    ~cplu1
 5. ~5r6p- & 5mp-- & 5btop &~5diro &~rnear & rrp2- &~rnea8 &~rneac &
     cwksa
 6.  srfil &~spran & 5btop &~5diro &~rnear &~rrp2- &~rpsq- &~rneap &
    ~rnea8 & rneac & cwksa &~cahea &~cplu1 &~cplu2
 7.  srfil &~spra7 &~5r6p- & 5btop &~5diro &~rnear & rpsq- & rneac
 8.  srfil &~spran & 5btop &~5diro &~mpmov &~mdirb &~rnear &~rrp2- &
     rnea8 & rneac & cwksa &~cahea &~cplu1 &~cplu2
 9. ~sint- &~spra7 &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp2- &
    ~mmp1- & rpsq- & cahea
10.  srfil &~spran &~spra7 &~5r6p- & 5btop &~5diro &~mpmov &~mmp1- &
    ~mdirb &~mnear &~rnear &~rrp2- & rpsq- & rneap &~rnea8 &~rneac
11.  srfil &~spran & mpmov &~rnea8 & rneac &~cplu1 &~cplu2
12. ~sint- &~spran &~spra7 &~5r6p- &~5diro &~mpmov &~mmp2- &~mmp1- &
    ~mdiro &~rrp2- &~ccrit & cplu2
13.  srfil &~spra7 &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp2- &
     ~mmp1- &~many- &~mnear &~rrp2- &~rrp1- &~rnea8 &~rneac &~ccrit &
     ~cwksa &~cahea &~cplu1 & cplu2
14. ~srfil &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp1- & mnear &
    ~rnear &~rrp2- &~ccrit &~cwksa &~cahea & cplu2
15. ~srfil &~spra7 &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp1- &
    ~rrp2- & rneap &~cwksa &~cahea & cplu2
16.  srfil &~rrp2- & rnea8 &~ccrit & cplu2
17.  srfil &~rnear &~rrp2- &~rnea8 & rneac &~ccrit & cplu2
18. ~srfil &~spran &~spra7 &~5mp-- &~mpmov &~mmp1- &~rstal &~rnear &
    ~rrp2- &~rrp1- & rneap &~cwksa &~cahea &~cplu1 & cplu2
19. ~spran &~mpmov &~mmp1- & rnear &~rrp2- &~rrp1- & rneap & rneac &
    ~cahea &~cplu1 & cplu2
20.  srfil &~5mp-- & 5btop &~5diro &~mmp2- &~mmp1- &~rnear &~rneac &
     ccrit & cwksa &~cahea & cplu2
21. ~sint- &~srfil &~spran &~spra7 & 5btop &~5diro &~mpmov &~mmp2- &
    ~mmp1- &~rrp2- &~rrp1- & rneap &~rnea8 &~rneac &~cplu1 & cplu2
22.  srfil &~spra7 &~5diro &~mpmov &~mmp2- &~mdirb &~rrp2- &~rrp1- &
     ccrit &~cwksa & cplu2
23. ~sint- &~spran &~5r6p- &~5diro &~mpmov &~mmp2- &~mmp1- &~mdiro &
    ~mdirb &~rrp2- &~cwksa & cplu1
24.  srfil &~spran &~spra7 &~5diro &~many- &~mdiro &~mdirb &~mnear &
    ~rrp2- & rnea8 &~cwksa & cplu2
25.  srfil &~rrp2- & rneac &~cahea & cplu1
26.  srfil & 5btop &~5diro &~mmp1- &~rnear &~rneap &~ccrit &~cahea &
     cplu1
27.  srfil &~spra7 &~5r6p- &~mpmov &~mmp2- &~mmp1- &~many- &~rnear &
    ~rrp2- & rnea8 &~cwksa & cplu2
```

Figure 3.12 (ctd.)

* * * Cover of Class 2 (LOST) * * *

1. ~cpat2 &~cwksa &~cahea &~cplu2 &~cimmt
2. ~srfil &~spra7 &~5btop & mmp1- &~mnear &~cimmt
3. ~srfil &~spra7 & mpmov &~rrp2- &~cpat2 &~cimmt
4. srfil &~spra7 &~rstal &~rnear &~rrp2- &~rnea8 &~rneac &~cahea & ~cimmt
5. ~srfil & 5mp-- & mmp1- &~rrp2- &~cpat2 &~cplu1 &~cimmt
6. ~spran & spra7 & 5mp-- &~rrp2- &~rneac &~cplu1 &~cimmt
7. ~srfil & spran &~spra7 &~5btop &~cpat2 &~cimmt
8. ~spran &~spra7 & 5mp-- & mmp2- &~rrp2- &~rneap &~rneac & cahea & ~cimmt
9. ~srfil &~spra7 & 5mp-- & mmp2- &~rrp2- &~cpat2 &~cimmt
10. ~spran &~spra7 & mpmov &~rrp2- &~rneac &~ccrit &~cimmt
11. ~srfil & spra7 &~5btop &~cpat2 &~cplu1 &~cimmt
12. ~srfil & spran &~spra7 & 5r6p- &~rrp2- &~cimmt
13. ~srfil & spran &~spra7 & 5diro &~rrp2- &~cimmt
14. ~srfil &~spra7 &~5btop & mdiro &~cimmt
15. sint- &~srfil &~spran &~many- &~rnear &~ccrit &~cwksa &~cplu1
16. ~rnear &~rneap & ccrit &~cwksa &~cplu1 & cplu2
17. sint- &~srfil &~spran &~rneac &~ccrit &~cwksa &~cplu1
18. ~spran & mnear & rrp1- &~rnea8 &~rneac &~ccrit &~cwksa &~cplu1
19. ~spran &~spra7 & mpmov &~rrp2- &~rneac & cahea &~cimmt
20. ~spra7 & 5mp-- & 5btop & mmp1- &~rrp2- &~rneac &~ccrit &~cimmt
21. ~srfil &~spra7 & 5mp-- &~mmp2- & mmp1- &~rrp2- &~cimmt
22. ~srfil & spran & 5mp-- & mdirb &~rrp2- &~cimmt
23. ~srfil & spra7 & 5mp-- & mdirb &~rrp2- & cplu1 &~cimmt
24. ~srfil &~spran &~spra7 &~mmp2- &~mmp1- &~mdirb &~rnear &~rnea8 & ~rneac &~ccrit & cwksa &~cahea &~cimmt
25. ~srfil &~spran &~spra7 & mpmov & rnear &~cimmt

Figure 3.13. Experiment 2 Comparison Table.

	initial pass on SAMPLE		second pass on 500+			complete runs		
	ID-3	AQ11P	ID-3 no-refinement	AQ11P	AQ11P refinement	ID-3	AQ11P single expansion	
conjunctive subconcepts								
drawn	19	8	16	17	32	22	27	
lost	16	14	20	17	24	24	25	
total vs. avg.	35	11	36	17	28	46	26	
avg. subconcept size								
drawn	6.4	4.2	8.3	4.6	8.6	8.5	11.1	
lost	7.1	4.4	8.6	4.3	6.5	9.0	7	
total	6.7	4.3	8.5	4.4	7.7	8.8	9.1	
max. subconcept size								
drawn	12	7	12	9	12	14	20	
lost	12	10	12	7	9	14	13	
total	12	10	12	9	12	14	20	
min. subconcept size								
drawn	3	1	1	1	6	1	1	
lost	3	2	5	2	5	3	5	
total	3	1	1	1	5	1	1	
number of different features tested	23	21,25	28	27,26	29,30	28	31,29	
% correctly classified	96.3	98	99.5	99.5	99.8	100	100	
rule generation time	1.8	15.3	3.7	50.3	18.2	11.2	69.4	(cpu seconds)
rule generation storage	~23000	30155	~23000	30375	30375	23062	37799	(# of 60 bit words)
figure 3. ·	6	7	8	9	10	11	12	

Results 2 : The results of this experiment are summarized in fig 3.13. In the first two passes (col.s 1-4) we again see AQ11P requires significantly more space and much more time to produce its decision rules than ID-3. When the covers produced in the initial pass are used as "input hypotheses" in pass 2, computation time is sharply reduced (col.s 4,5). AQ11P's decision rules require fewer conjunctions and fewer tests per conjunction than those generated by ID-3, but the reduction in computation time gained using rule-refinement (in col. 4) came at a cost of increased rule complexity. The last two columns compare two complete rules. Again, AQ11P required more resource expenditure. Notice that the use of input hypotheses has reduced the computation time to get complete rules for the whole data set from 325 to 70 seconds. Again, however, the "cheaper" covers are much more complicated than those obtained without initial hypotheses, (col. 7).

3.5 Experiment 3 : Rule Refinement

In this (final) experiment, an attempt is made to derive simple covers (using AQ11P) for comparison with the best ID-3 decision trees for KPK. The simplest KPK-BTM decision tree was produced by Niblett and Shapiro, (figure 3.14).

To determine whether the AQ "incremental hypothesis generation" method can be used to produce simple rules by rule-refinement, AQ11P is given covers that are already very good (simple in the sense of "comprised of a small number of small conjunctions" and complete in the sense of "correct for all feature vectors"). New covers are generated for the whole set of feature vectors.

Method 3 : AQ11P was run on all 2411 examples using the simple and complete rules of figure 3.15 as initial hypotheses. The resulting rules are shown in figure 3.16.

Figure 3.14. Best ID-3 Decision Tree for KPK.

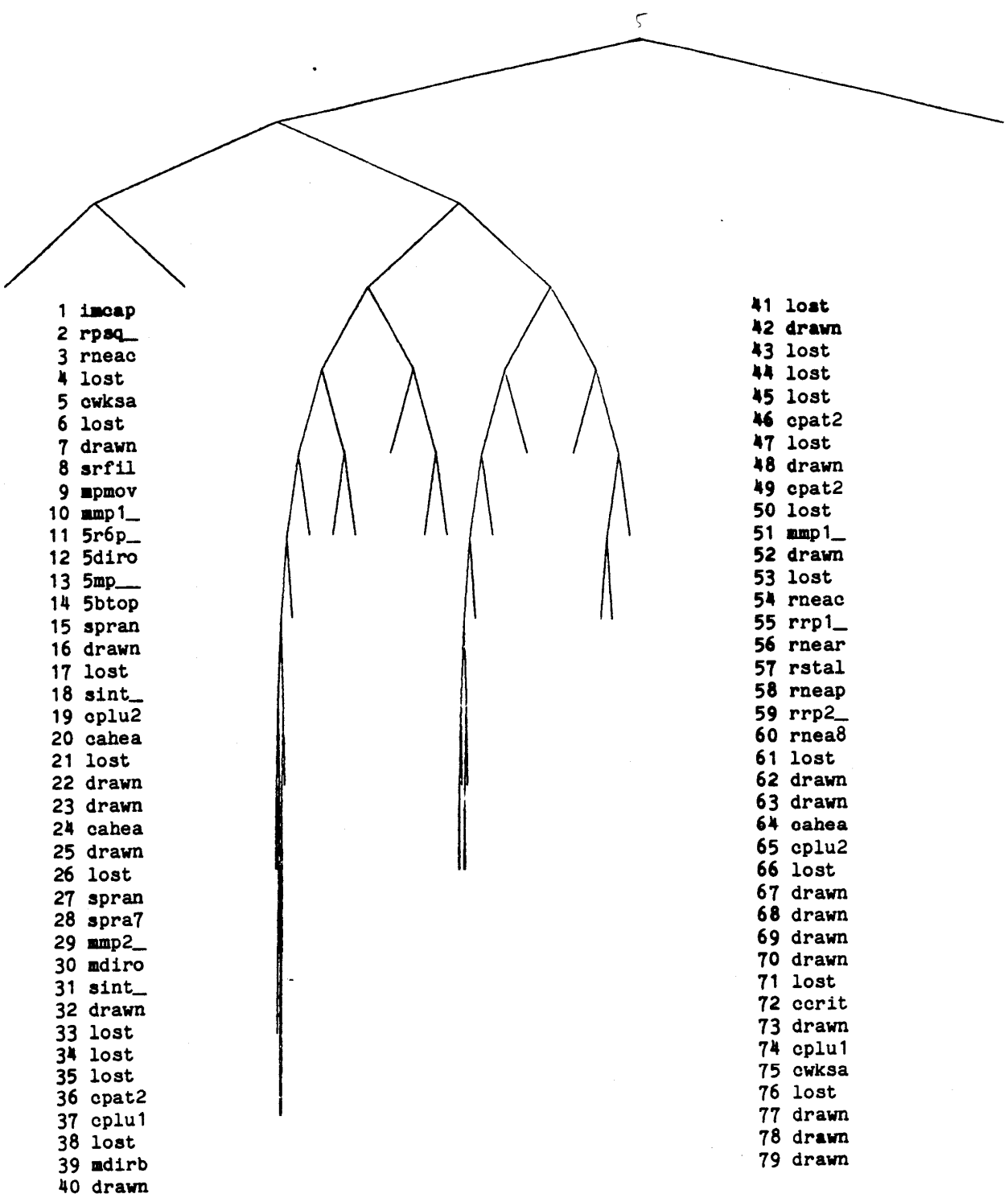


Figure 3.15. Rules to be refined by AQ11P.

* * * Cover of Class 1 (DRAWN) * * *

```
1. ~cimmt
2. ~sint- &~spra7 & mnear & rnear &~rneac &~cplu1
3. ~mmp1- & cpat2
4.  srfil & rstal
5.  srfil & rrp2-
6.  srfil & rneac & cwksa
7. ~sint- &~5r6p- & 5btop &~5diro &~mpmov &~mmp2- &~mmp1- & cahea
8.  srfil &~mpmov &~mnear & rpsq- & rneap
9. ~sint- &~spran &~spra7 &~mpmov &~mmp2- &~mmp1- &~mdiro & cplu2
10. ~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~ccrit &~cahea & cplu2
11.  srfil &~spra7 &~5r6p- &~many- & rpsq- & rneac
12.  5btop & many- & rpsq- &~rneap
13.  srfil &~spran &~mnear & rpsq- & rnea8
14. ~spran &~5btop & many- &~ccrit & cplu2
15. ~5r6p- & 5btop &~mmp1- &~mdirb & mnear & cplu1
16.  srfil & 5r6p- & cplu1
17. ~5mp-- & 5btop & mdirb & ccrit
```

* * * Cover of Class 2 (LOST) * * *

```
 1. ~cpat2 &~cwksa &~cahea &~cplu2 &~cimmt
 2. ~srfil & 5mp-- & mmp1- &~mnear &~cimmt
 3.  mpmov &~rstal &~rrp2- &~rneac &~cpat2 &~cimmt
 4.  5mp-- &~mpmov & mmp1- &~rnea8 &~rneac &~cimmt
 5. ~srfil & spran &~5btop &~cpat2 &~cimmt
 6.  mmp2- &~rneac &~cwksa &~cplu1 & cplu2 &~cimmt
 7. ~srfil & spra7 &~5btop &~cpat2 &~cimmt
 8. ~srfil & spran & 5r6p- &~cimmt
 9. ~srfil & 5mp-- & mmp1- &~cpat2 &~cimmt
10. ~srfil & spran & 5diro &~cimmt
11. ~srfil &~spran & mmp2- &~cimmt
12. ~srfil &~5btop & mdiro &~cimmt
13.  sint- &~srfil &~many- &~cimmt
14.  5r6p- & mmp2- & ccrit &~cimmt
15.  spra7 &~rnear & rpsq- & ccrit &~cahea &~cplu1 &~cimmt
16.  spran &~5btop & many- & ccrit &~cimmt
17. ~srfil &~spran & mpmov &~cimmt
18.  sint- &~rneac &~cimmt
19. ~srfil & spran & 5mp-- & 5btop &~cahea &~cimmt
20. ~srfil & spra7 & 5mp-- & many- &~cahea & cplu1 &~cimmt
21. ~mpmov & mmp2- &~rneap &~rneac & cahea &~cimmt
```

Figure 3.16. Results of attempted refinement

* * * Cover of Class 1 (DRAWN) * * *

1. cimmt
2. ~sint- & spran &~spra7 &~5r6p- &~5btop &~5diro &~mpmov &~many- & ~mdiro & mnear & rneap & rnea8 & cpat2 &~cplu2
3. ~sint- & spran &~spra7 &~5r6p- &~5btop &~5diro &~mmp1- &~mdiro & cpat2 &~ccrit &~cahea &~cplu2
4. ~sint- & srfil &~spra7 &~5r6p- & 5btop & mpmov & rstal & cwksa
5. ~sint- & srfil &~spra7 &~5r6p- & mpmov &~many- & rrp2- & cwksa
6. srfil &~spran &~mmp2- &~rpsq- & rneac & cwksa
7. srfil &~spra7 &~5r6p- &~many- & rneac & cwksa
8. ~sint- &~5r6p- & 5btop &~5diro &~mpmov &~mmp2- &~mmp1- & cahea
9. ~sint- & srfil &~5r6p- &~mpmov &~mmp1- &~many- & rneap & cahea & ~cplu2
10. ~sint- &~spran &~spra7 &~mpmov &~mmp2- &~mmp1- &~mdiro & cplu2
11. ~sint- &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp2- &~ccrit & cplu2
12. srfil &~spra7 &~5r6p- &~many- & rneac & cplu2
13. srfil & rneac &~ccrit & cplu2
14. ~spra7 &~5r6p- &~5mp-- & 5btop &~5diro &~mpmov &~mmp2- & many- & rneac & cplu2
15. ~spran &~spra7 &~5mp-- &~mpmov &~mmp2- & many- &~mdiro & rneac & cplu2
16. ~sint- &~spran &~5btop &~mpmov &~mmp2- &~mmp1- &~mdiro & rnear & rneac & cpat2 &~cplu1
17. ~sint- & srfil &~spran &~5r6p- &~mpmov & rnea8 &~cahea & cplu1 & cplu2
18. ~sint- &~spran & 5btop &~mpmov &~mmp2- &~mmp1- &~many- & cplu1 & cplu2
19. ~sint- & srfil &~spra7 &~5mp-- & 5btop &~mpmov &~mmp2- & cplu2
20. ~sint- & srfil &~5r6p- & 5btop &~mpmov &~mmp1- & rneap & cplu1 & cplu2

* * * Cover of Class 2 (LOST) * * *

1. ~cpat2 &~cwksa &~cahea &~cplu2 &~cimmt
2. ~srfil & 5mp-- & mmp1- &~many- &~mdirb &~mnear &~rrp2- &~cahea
3. ~rpsq- &~rneac &~cahea &~cplu2 &~cimmt
4. ~srfil &~cpat2 &~cahea &~cplu2 &~cimmt
5. srfil &~spra7 &~mnear &~rstal &~rnear &~rrp2- &~rnea8 &~rneac & ~cahea &~cimmt
6. ~srfil & 5mp-- &~5btop & mmp1- &~many- &~mdirb &~mnear &~rnear
7. ~srfil & 5mp-- & mmp2- &~many- &~mdirb &~mnear &~rrp2- &~cpat2 & ~ccrit &~cimmt
8. srfil &~mdiro &~mdirb &~mnear &~rnear &~rrp2- &~rneap &~rnea8 & ~rneac &~cwksa &~cimmt
9. ~srfil & 5mp-- &~5btop &~mnear &~rrp2- &~rneac &~cpat2
10. srfil &~spra7 &~5btop &~rnear &~rrp2- &~rneap &~cplu2
11. ~srfil & spran &~5btop &~mdiro &~mdirb &~rrp2- &~cpat2 &~cwksa
12. ~srfil &~spran & mpmov &~mdirb &~rnea8 &~rneac &~ccrit &~cimmt
13. ~sint- &~srfil &~spran & spra7 &~5btop &~mmp2- &~mmp1- &~many- & ~mdiro &~mdirb &~rstal &~rnear &~rrp2- &~cpat2 &~cwksa &~cahea & ~cplu1
14. ~spran &~5mp-- & rrp1- &~cwksa &~cplu1 &~cimmt
15. ~sint- &~srfil &~spran &~5btop &~many- &~mdiro &~mdirb & rneap & ~rneac &~cpat2 &~cahea &~cplu1
16. ~srfil & spran & 5mp-- &~many- &~mnear &~rrp2- & rneac &~cwksa & ~cimmt
17. ~srfil & 5mp-- &~mmp2- & mmp1- &~many- &~mdirb & mnear &~rnear & ~rrp2- &~ccrit
18. 5diro &~rrp2- & rneap &~ccrit &~cwksa
19. ~srfil & 5r6p- & mnear &~rrp2- &~ccrit &~cwksa
20. ~srfil &~spran & 5mp-- &~many- & mdiro &~mdirb &~rnear &~rrp2- & ~ccrit &~cwksa
21. sint- &~srfil &~many- &~rstal &~rnear &~rrp2- &~cpat2 &~cwksa & ~cahea &~cplu1
22. ~srfil &~spra7 &~mmp2- &~mmp1- &~many- &~mdiro &~rnear &~rrp2- & ccrit &~cahea &~cimmt
23. ~srfil &~spran &~5mp-- & mpmov &~cwksa &~cplu1 &~cimmt
24. ~sint- & spran &~5diro & mmp2- &~mmp1- &~many- &~mdiro &~mdirb & ~rstal &~rnear &~rrp2- &~rneap &~rnea8 &~cpat2 &~cwksa &~cahea & ~cplu1

25. ~sint- & srfil &~spran & spra7 &~5btop &~mdiro &~rstal &~rnear & ~rrp2- &~rneap &~cpat2 &~cwksa &~cahea &~cplu1
26. ~sint- &~spran &~mmp2- &~mmp1- &~many- &~mdiro &~mdirb & mnear & ~rstal &~rnear &~rrp2- &~rneap &~cpat2 & ccrit &~cwksa &~cahea & ~cplu1
27. ~sint- & spran &~spra7 &~5mp-- &~5btop & mdirb &~mnear &~rstal & ~rrp2- &~cpat2 &~cwksa &~cahea &~cplu1
28. ~spran & 5mp-- & many- & mdirb &~rrp2- & rneap &~cpat2 &~cwksa & ~cahea
29. ~srfil &~spran &~spra7 &~5r6p- &~5mp-- &~mmp2- &~mmp1- &~mdirb & ~rrp2- & rneap &~rneac & ccrit &~cahea &~cimmt
30. sint- &~srfil &~spran &~rneac &~ccrit &~cwksa &~cplu1
31. 5mp-- &~mdirb &~mnear &~rnear &~rrp2- &~rnea8 &~rneac &~ccrit & ~cimmt
32. ~srfil &~spran & 5btop &~many- &~mdirb &~mnear &~rnear &~rnea8 & rneac &~ccrit &~cwksa &~cplu1 &~cimmt
33. ~srfil &~spran &~5btop & mmp1- &~many- &~mdirb & mnear &~rrp2- & ~ccrit &~cwksa
34. sint- &~srfil &~spran &~many- &~mdirb &~rnear &~ccrit &~cwksa & ~cplu1
35. ~srfil &~spran & 5mp-- & mmp2- &~many- &~mdiro &~mdirb &~mnear & ~rrp2- & ccrit &~cwksa &~cimmt
36. ~srfil & spran & 5mp-- &~cpat2 &~cwksa &~cplu1 &~cimmt
37. ~srfil & spran &~spra7 &~mpmov & mmp1- &~many- &~mdirb &~rrp2- & ~rneap &~cwksa &~cahea
38. ~spran & mpmov &~mmp2- &~mdirb &~rrp2- &~ccrit &~cwksa &~cimmt
39. ~srfil &~spra7 & 5btop &~many- & mdirb & mnear &~rnear &~rrp2- & ~ccrit &~cwksa &~cahea & cplu1
40. ~srfil & spran &~mpmov &~mmp1- &~many- &~mdiro &~rrp2- &~rneap & ccrit &~cahea
41. ~spra7 &~5btop &~mpmov &~many- &~mdirb & mnear &~rnear &~rrp2- & rneap & ccrit &~cwksa &~cahea
42. ~srfil & 5mp-- &~mmp1- &~many- &~mdirb &~mnear &~rrp2- & ccrit & ~cahea &~cimmt
43. ~spran & 5mp-- &~5btop &~mpmov &~mmp2- &~mmp1- &~many- & mdiro & ~mdirb &~mnear & rnear &~rrp2- &~rneap &~ccrit &~cwksa
44. ~srfil &~spran & 5mp-- &~many- &~mdirb &~mnear & rnear &~rrp2- & rneap & rneac &~ccrit &~cwksa

Comment 3: The resulting rules are (by inspection) more complicated than the original rules. AQ's rule-refinement mapped simple input covers into a more complicated set even when the input covers correctly classified 100% of the target feature vectors. Later an explanation is offered for this surprising behavior. Since simpler rules cannot be constructed by using the rule-refinement mechanism in AQ11P, we now compare the simplest decision rules for KPK generated in all previous experiments.

Method 3 : We examine the simplest covers generated by AQ11P (in the base comparison of Experiment One) together with Niblett and Shapiro's best ID-3 decision tree.

Result 3 : The simplest decision rules for KPK-BTM derived by AQ11P and ID-3 appear in figures 3.3 and 3.14. The table in figure 3.17 compares various features of these rules.

Figure 3.17. Experiment 3 Comparison Table.

	ID-3 DECISION TREE	AQ11P COVERS
no. of conjunctive subconcepts		
drawn	20	17
lost	20	17
total vs. avg	40	17
avg. conjunction size		
drawn	8.4	4.6
lost	8.6	4.1
total	8.5	4.3
max. conjunction size		
drawn	14	8
lost	14	8
total	14	8
min. conjunction size		
drawn	2	1
lost	4	2
total	2	1
number of different features tested	28	29
% correctly classified	100%	100%

4 Discussion

Previous sections summarized two induction methods (AQ and IP) and compared their implementations (programs AQ11P and ID-3) via three experiments. This section carries the examination of the forms and methods of construction of decision rules further, and then discusses the implementations.

4.1 Forms of Decision Rules

Automated induction systems such as AQ11P and ID-3, which can derive general decision rules from examples of expert's decisions promise to lift some of the burden of knowledge-acquisition from the shoulders of knowledge-engineers constructing rule-based expert systems. A key precondition for fulfilling this promise is that the computer produced rules should be humanly comprehensible. In this section, a simple measure intended to reflect "cost of understanding" is defined on a class of decision structures which includes both decision trees and covers. Next, the value of this measure is determined for a number of examples. Finally, the cost of classification using trees and covers is discussed.

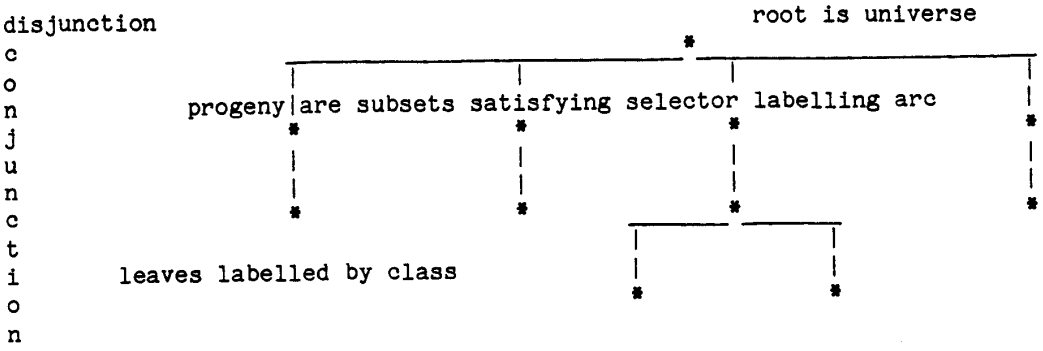
4.1.1 Subtlety

If automated induction systems are to contribute to the developement of rule-bases for expert systems, their products must be humanly comprehensible. But how does one objectively compare different decision structures with respect to comprehensibility?

Start by viewing covers and trees as special cases of a (slightly odd) class of discrimination nets [Charniak et al. 80] or AND/OR graphs [NILSSON 80], which we will call "selector trees."

As with decision trees, the root of a selector tree corresponds to the entire set of objects to be classified, but edges emanating from a node may be labelled with any selector. The subsets corresponding to progeny of a node are the elements of the node's set which satisfy the corresponding selector. They need not be disjoint. The selector tree is not restricted to splitting on just one feature, nor is it required to use all values of a feature. Class labels appear exactly once on each path in a selector tree, but not necessarily at the leaves.

In this paper, selector trees will be pictured:

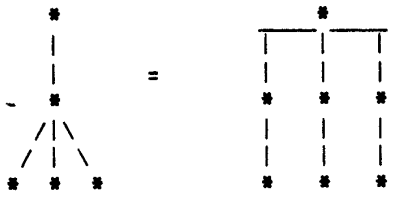


Covers are selector trees whose leaves are all labelled by the same class and whose branches never bifurcate after the split from the root.

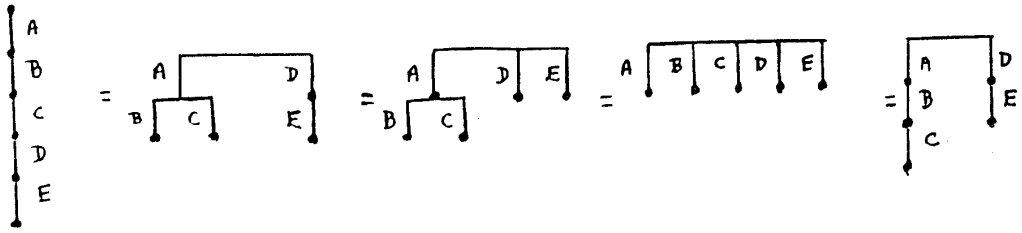
Selector trees define decision relations by associating an object with the set of class tags labelling paths which correspond to conjunctions of selectors true for that object.

Now a subtlety measure can be assigned to each node in the selector tree version of the decision rule. This measure will simply be the depth of the node. The intuition is that the number of selectors in the individual conjunctive concepts corresponding to nodes is a reasonable measure of the complexity of the conjunctive concepts. With this convention, the average depth of the leaves corresponds to the average complexity of the conjunctive subconcepts in the decision rule.

Together with the total number of leaves (paths), this gives some feeling for the subtlety of the rule. The product of the number of paths and the average path length was considered as a measure of rule complexity, but this product has the disadvantage that the node-sharing inherent in decision trees is not appreciated. Nodes which appear in more than one path are counted more than once by this measure, so it does not reflect our intuitive feeling that selector trees like the following should be assigned different complexities.



Another candidate for a measure of rule complexity is the total number of selectors, (counting each selector which appears in several paths as many times as it appears). This measure appreciates node-sharing, but unfortunately, doesn't charge for added logical complexity. For example, it considers all the following to be equally simple:



A measure which gives credit for node-sharing and which also charges for logical complexity can be had by considering selector trees as strings in the language defined by the following grammar.

s-->(or s^)	s'-->(or s#)
s-->(and s^)	s'-->(and s#)
s^--> s s*	s'-->selector
s*--> s s*	s#--> s' s$
s*--> s	s$--> s' s$
s--> s' class	s$--> s'

Where s is the initial symbol and s,s^,s*,s',s and s$ are nonterminals.

The strings in this language are the obvious prefix logical expressions for selector trees. The length of an expression (not counting parentheses, but counting selectors, logical connectives and class tags) appears to be a reasonable measure of the subtlety of the correponding decision rule.

Given an expression it is easy to compute its cost recursively:

cost(selector)=1,

cost((and s1...sn))=cost((or s1...sn))= 1+cost(s1)+...+cost(sn),

cost(s class)=1+cost(s).

It is also easy to compute the cost of a selector tree directly from the pictorial form: -

cost(selector-tree)= number of selector appearances + number of class tags present + (for each split, 1+ the number of subtrees consisting of more than one selector)

The expressions for the selector trees in the previous diagram and the corresponding values of this final measure of "cost of understanding" are (assuming the root of each selector tree is labelled with a class tag):

Expression	Cost
(and a b c d e)	7
(or (and a (or b c)) (and d e))	10
(or (and a (or b c)) d e)	9
(or a b c d e)	7
(or (and a b c) (and d e))	9

This measure still has weaknesses. For example it is asymmetrical with respect to AND and OR. Worse, (as R.S. Michalski has pointed out), it is too linear to be a good measure of the human-comprehensibility of decision structures. A more realistic "cost of understanding" measure would be highly nonlinear, would probably charge less for logical connectives than for selectors, and would certainly charge more for some selectors than for others to reflect differences in the comprehensibility of the features. Nevertheless, for lack of a better instrument, perhaps based on psychological studies, we commit ourselves to using this crude, but hopefully informative, measure.

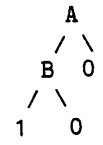
4.1.2 Trees vs. Covers

In this section the subtlety measure just introduced is used to compare decision tree and cover representations of some simple logical concepts taken from [Hunt et.al. 66]. Then the simplest rules developed by ID-3 and AQ11P for the KPK-BTM chess endgame problem are investigated.

1) Basic Logical Concepts (problem set 1 in HUNT)

In these examples, the left hand arc is implicitely labelled [feature-value=1] or TRUE; the right arc contrarily. There are two covers corresponding to each decision tree since the example concepts are all boolean. Only one cover is required for a decision rule in this case, so the second is encased in parentheses.

a) A and B:

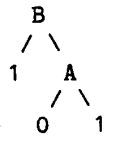


$AB \rightarrow 1$

($\bar{A} \backslash / \bar{B} \rightarrow 0$)

b) A or B: similar to a.

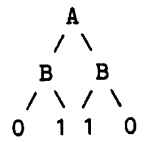
c) A implies B:



$B \rightarrow 1$

($A\bar{B} \rightarrow 0$)

d) A xor B:



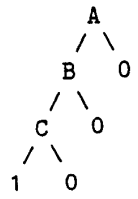
$A\bar{B} \backslash / \bar{A}B \rightarrow 1$

($AB \backslash / \bar{A}\bar{B} \rightarrow 0$)

e) A equivalent B: similar to d.

2) Some 3 feature concepts (problem set 2 from HUNT)

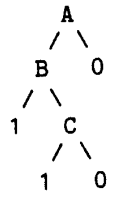
a) A and B and C



ABC-->1

($\bar{A}\backslash/\bar{B}\backslash/\bar{C}$-->0)

b) (A and B or C)

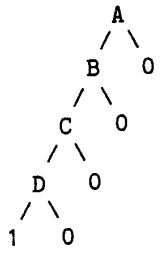


AB\/AC-->1

($\bar{A}\backslash/\overline{BC}$-->0)

3) Some 4 feature concepts.

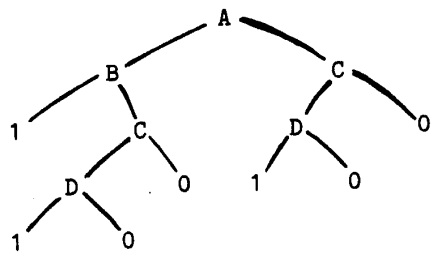
a) A and B and C and D



ABCD-->1

($\overline{A}\backslash/\overline{B}\backslash/\overline{C}\backslash/\overline{D}$-->0)

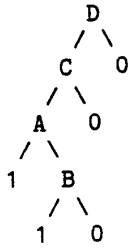
b) (A and B) or (C and D)



AB\/CD-->1

($\overline{AC}\backslash/\overline{AD}\backslash/\overline{BC}\backslash/\overline{BD}$-->0)

c) (A or B) and C and D



ACD\/BCD-->1

($\overline{C}\backslash/\overline{D}\backslash/\overline{AB}$-->0)

This table compares the decision rules just presented and the simplest decision rules for KPK (from experiment three).

	NUMBER OF PATHS			*	AVG PATH LENGTH			*	COST
	0	1	tot.vs.avg.		0	1	tot.		
1a)									
tree	2	1	3		1.5	2	1.67		10
covers	2	1	3/2		1	2	1.33		4,4, avg=4
1c)									
tree	1	2	3		2	1.5	1.67		10
covers	1	2	3/2		2	1.5	1.67		6,4, avg=5
1d)									
tree	2	2	4		2	2	2		15
covers	2	2	2		2	2	2		8,8, avg=8
2a)									
tree	3	1	4		2	3	2.25		15
covers	3	1	2		1	3	1.5		5,5, avg=5
2b)									
tree	2	2	4		2	2.5	2.25		15
covers	2	2	2		1.5	2	1.75		8,6, avg=7
3a)									
tree	4	1	5		2.5	4	2.8		20
covers	4	1	5/2		1	4	1.6		6, 6, avg= 6
3b)									
tree	4	3	7		3	3	3		30
covers	4	2	3		2	2	2		8,14, avg=12
3c)									
tree	3	2	5		2.3	3.5	2.8		20
covers	3	2	5/2		3	1.3	2		10,7, avg=8.5
KPK)									
tree	20	20	40		8.35	8.55	8.45		194
covers	17	17	17		4.59	4.06	4.33		97,88 avg=92.5

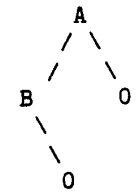
In the table the average number of paths of the two covers is compared to the total number of paths in the corresponding tree. This is done because only one cover is required for a decision rule, but paths for both classes must be present in the tree. Notice that the average number of paths in the covers is never more than half the average number of paths in the trees, and that the average length of the paths in covers is also smaller.

According to our simple cost-of-understanding measure, decision rules expressed as decision trees are consistently almost twice as complicated as their cover counterparts. Why is this so?

First, only n-1 covers are needed to decide membership among n classes when one assumes that any object not covered is in the remaining class. Similarly, in a decision tree, only the paths leading to n-1 of n classes are really needed under this assumption. When n=2, as in all the examples, this advantage of covers over decision trees is maximal. About half the decision tree paths are probably unnecessary.

Second, (even if the above bias is taken into account and only the paths leading to n-1 of n classes are considered), the one-feature-at-a-time requirement on splitting would still lead to excess complexity in the decision trees.

For example: extracting the 0-paths from the decision tree for the concept A or B yields



which, mapping paths into terms in a cover, is equivalent to

$\bar{A} \rightarrow 0$

$A\bar{B} \rightarrow 0$

Since the decision is not sensitive to the context A in the second term, this is equivalent to the cover

$$\overline{A} \text{ -->} 0$$

$$\overline{B} \text{ -->} 0$$

Deletion of the unshared parts of paths leading to one of the n classes does not lead to much of an improvement , since only one selector of each such path is not shared with paths leading to other classes. Perhaps there are ways of mapping decision trees to simpler selector trees, (eg. by applying absorption to cases like the one above).

4.1.3 Rule Execution (Pattern Recognition)

This section concludes the discussion of the different decision structures used in the ID and AQ inductive learning methodologies with a few remarks on rule execution. The "execution" of a (correct) decision rule on an example yields the name of the class to which that example belongs. In this discussion it is assumed that the pattern to be classified has already been coded as a feature vector.

Sequential test procedures look at features of an example, using the values to steer down a unique path in a decision tree. The class name label of the leaf at the end of that path is returned as the result of execution.

Execution of covers on sequential machines is less efficient, because it may be necessary to try more than one conjunctive term in a cover, and, in general, more than one cover, before a determination of class can be made. Some redundant feature testing can be eliminated by binding together complexes with common selectors (at the cost of an increase in the complexity of the rule evaluator). See the section suggesting improvements for decision rules. If parallel machines are available, all the terms of all covers can be evaluated independently. In fact, the individual selectors in each term can be tested simultaneously. Because of their logical sum-of-products form, large fixed covers can be implemented in Programmable Logic Arrays. When covers are evaluated in AQ11P on the Control Data Corporation CYBER computer systems, all the selectors of each term are evaluated "in parallel" via 1 or 2 machine instructions, but the terms are tested one at a time.

4.2 Construction of Decision Rules

The main differences between the rule construction methods of AQ and IP are:

1. IP is top down: it partitions the feature space into cells, separating elements of different classes by looking only at relative frequencies of classes of objects in the cells. AQ is different: it looks at a positive example and computes a good cell around it which misses all the negative examples, etc.

2. IP has the "windowing" iterative expansion scheme: exceptions are added to the current window and a new rule is generated which takes care of them, etc. Usually the final sample yields a complete decision rule even though the sample is only a small subset of the target set. AQ lacks this, but it allows one to use initial hypotheses to help generate correct covers, where IP must always start from scratch. AQ also allows the input of arbitrary complexes as examples instead of just feature vectors.

3. The examination of both the methods and their implementations leads to the observation that AQ requires more computational resources than IP. On the other hand, the decision rules produced by AQ seem consistently less complicated than IP decision trees.

5 Suggestions

5.1 For Improved Rule Construction

IP: Why should new rules be started from scratch? It may be possible to cut production time (probably at a loss of final rule quality) by introducing some way for IP to build on its previous rule. If the number of exceptions is small, the new rule is not likely to be much different from the old one. Hunt discusses some schemes whereby only the bad paths in the tree are modified. [HUNT et al. 66]

Currently IP is susceptible to the "horizon effect" of Berliner, since it does not look ahead when deciding which feature to split on. It chooses the feature which gives the best immediate gain, with no regard for the difficulty of further splitting the resultant sets.

On the subject of split selection, a standard heuristic for constructing short sequential testing procedures is to balance the sizes of the split

sets produced [RIENGOLD et al 77]. It is possible (and may be profitable) to add this heuristic to IP.

AQ:

The expansion strategy currently implemented in AQ11P should be dropped. Perhaps AQ should be given the kind of incremental rule formation strategy proven by ID-3. This might enable AQ to tackle large sets of examples.

AQ's current rule-refinement method has the following technical problems:

1) Unnecessary computations are currently included in the method.
The incremental rule generation process of [MICHALSKI and LARSON] (summarized in section 2.2.2 of this report) can be simplified to -

A) for all i, determine missing examples:

$$E_i^A = E_i - H_i$$

B) for all i, determine new covers:

$$H_i' = \text{Cover}(E_i, [\bigcup_{k \neq i} H_k \cup E_k^A]).$$

Recall that rule-refinement was just a fast way of computing an approximation of

$$\text{Cover}\ (E_i, \bigcup_{k \neq i} E_k).$$

In fact, the method involved computing an approximation

$$\text{Cover}\ (E_i, \bigcup_{k \neq i} [(H_k - H_k^S) \cup E_k])$$

of the approximation

$$\text{Cover}\ (E_i, \bigcup_{k \neq i} [(H_k - (\bigcup_{\ell \neq k} E_{k\ell}^S)) \cup E_k])$$

It turns out that the latter approximation can be computed directly, however, since

$$\bigcup_{k \neq i} [(H_k - (\bigcup_{\ell \neq k} E_{k\ell}^S)) \cup E_k] \cup E_i = \bigcup_{k \neq i} [H_k \cup E_k] \cup E_i.$$

2) The present method of generating new covers from old ones tends to produce over-complicated results. The reason is probably: If the input covers are simple, they cover large volumes of feature space. To cover a class of examples against the others, AQ must produce complexes which miss a union of these unions of voluminous hyperboxes. This is the reason the resulting complexes are long: they have to be rather specific. Any given example can only expand to a small volume before it hits the covers of examples from the other classes. There are more complexes in the resulting covers because the complexes are forced to be small. The problem is that much of the space covered by simple initial hypotheses is really don't-care space and shouldn't really be off-limits. A solution is to shrink the input hypotheses (make them more specific) so that they fit the observed examples more closely before computing the new covers.

Surprising new evidence in support of this explanation has been supplied to the author by Kent Spackman and his fellow medical interns as a side effect of a course project in Spring of 1981. They observed oscillation of the complexity of rules produced by AQ11P. They were generating decision rules from sample medical diagnoses using many passes, with the rules produced on a sample used as input hypotheses for the next sample. The initial decision rules (produced with no initial hypotheses) were simple (and general), the next ones complex (ie. more specific), the next were simple, and so on alternating for several more iterations!

5.2 Suggestions for Improvement of the Decision Rules

The covers produced by AQ11P have many terms with common factors. If terms with a common factor are "bound" to a single copy of that factor, the number of selector-appearances is reduced. A LISP program has been written which constructs selector trees from covers in an effort to simplify them. Results of applying this (simple and non-optimal) binding scheme to the best KPK-BTM covers follow.

The bound KPK cover for the class of DRAWN board positions is:

```
(or
 (and (srfil=t)
      (or
       (and (rpsq_=t)
            (or
             (and (mnear=f)
                  (or (and (spran=f) (rnea8=t))
                      (and (mpmov=f) (rneap=t))))
             (and (spra7=f) (5r6p_=f) (anyop=f) (rneac=t))))
       ((rstal=t))
       ((rrp2_=t))
       (and (rneac=t) (cwksa=t))
       (and (5r6p_=t) (cplu1=t))))
 (and (5btop=t)
      (or
       (and (5r6p_=f)
            (or
             (and (mmp1_=f)
                  (or
                   (and (sint_=f)(5diro=f)(mpmov=f)(mmp2_=f)(cahea=t))
                   (and (mdirb=f) (mnear=t) (cplu1=t))))
             (and (5mp  =f)(5diro=f)(mpmov=f)(ccrit=f)(cahea=f)(cplu2=t))))
       (and (5mp__=f) (mdirb=t) (ccrit=t))
       (and (anyop=t) (rpsq_=t) (rneap=f))))
 (and (sint_=f) (spra7=f)
      (or (and (mnear=t) (rnear=t) (rneac=f) (cplu1=f))
          (and (spran=f)(mpmov=f)(mmp2_=f)(mmp1_=f)(mdiro=f)(cplu2=t)))))
 (and (spran=f) (5btop=f) (anyop=t) (ccrit=f) (cplu2=t))
 (and (mmp1_=f) (cpat2=t))
 ((cimmt=t)))
```

The expression for the bound LOST cover is:

```
(or
 (and (srfil=f)
      (or (and (mmp2_=f) (or (and (mpmov=t) (cimmt=f)) ((mmp1_=t))))
          (and (cpat2=f)
               (or (and (mmp2_=t) (cimmt=f)) (and (spran=t) (5btop=f))))
          (and (5mp__=t) (5btop=t) (mdirb=t) (cahea=f))
          (and (sint_1) (anyop=f))
          ((mdiro=t))
          ((5r6p_=t))
          (and (mmp1_=t) (mnear=f))))
 (and (rnear=f)
      (or
       (and (5mp__=t) (mpmov=f) (rneac=f)
            (or
             (and (mnear=f)(rneap=f)(rnea8=f)(cahea=t)
             (and (rstal=f) (cahea=f) (cimmt=f)))
       (and (srfil=t) (rneap=f) (ccrit=t) (cwksa=f) (cplu1=f))))
 (and (cpat2=f)
      (or
       (and (cimmt=f)
            (or (and (cwksa=f) (cahea=f) (cplu2=f))
                (and (mpmov=t) (rstal=f) (rrp2_=f) (rneac=f))))
       (and (spra7=t) (5btop=f))))
 (and (sint_t) (rrp1_=t))
 (and (5diro=t) (rneap=t)))
```

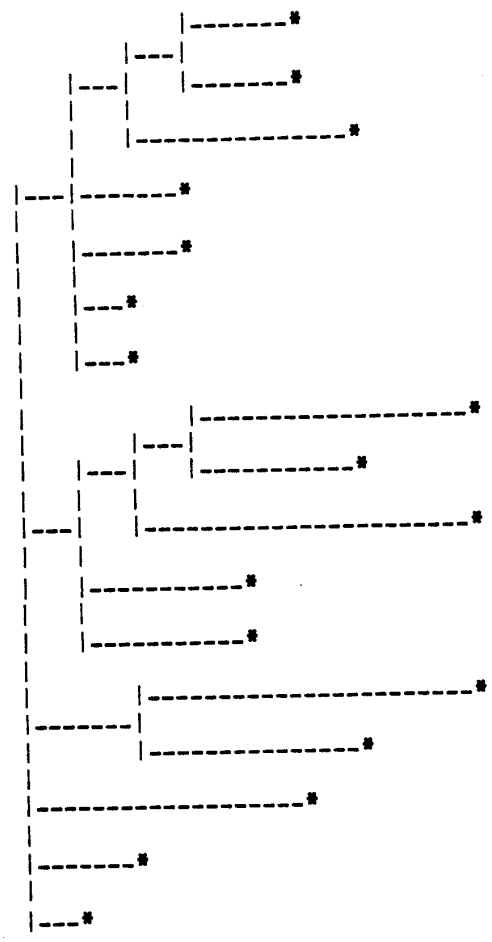
Table comparing bound covers with originals.

	selector appearances lost	drawn	* cost lost	drawn
before	69	78	88	97
after	51	60	81	90
relative improvement	26%	23%	8%	7%

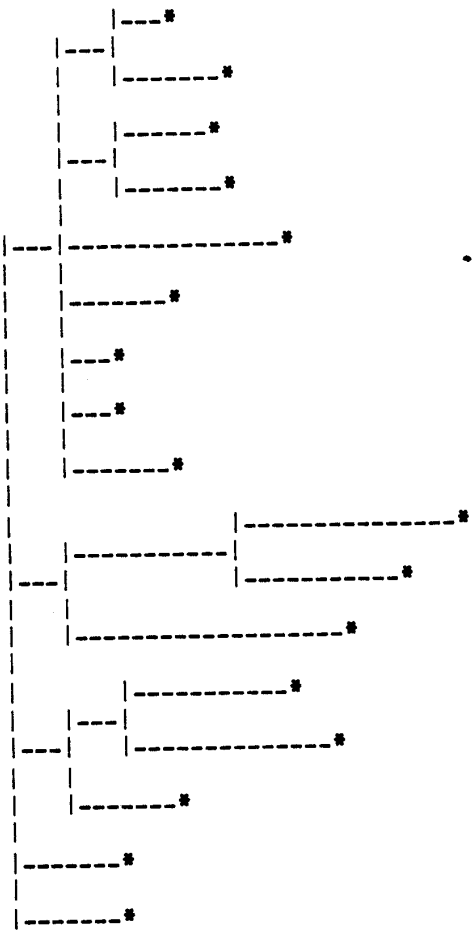
Notice that binding reduced the number of selector appearances by 1/4th, but this improvement was paid for with increased logical complexity, so that the overall cost reduction was less than 10%. Still, this gain is probably more significant than it appears, since a more realistic cost-of-understanding measure would be highly non-linear and would probably charge less for logical connectives than for selectors.

For an indication of the relative complexity of the bound covers with respect to the best decision tree produced by ID-3 for KPK, see the following figures.

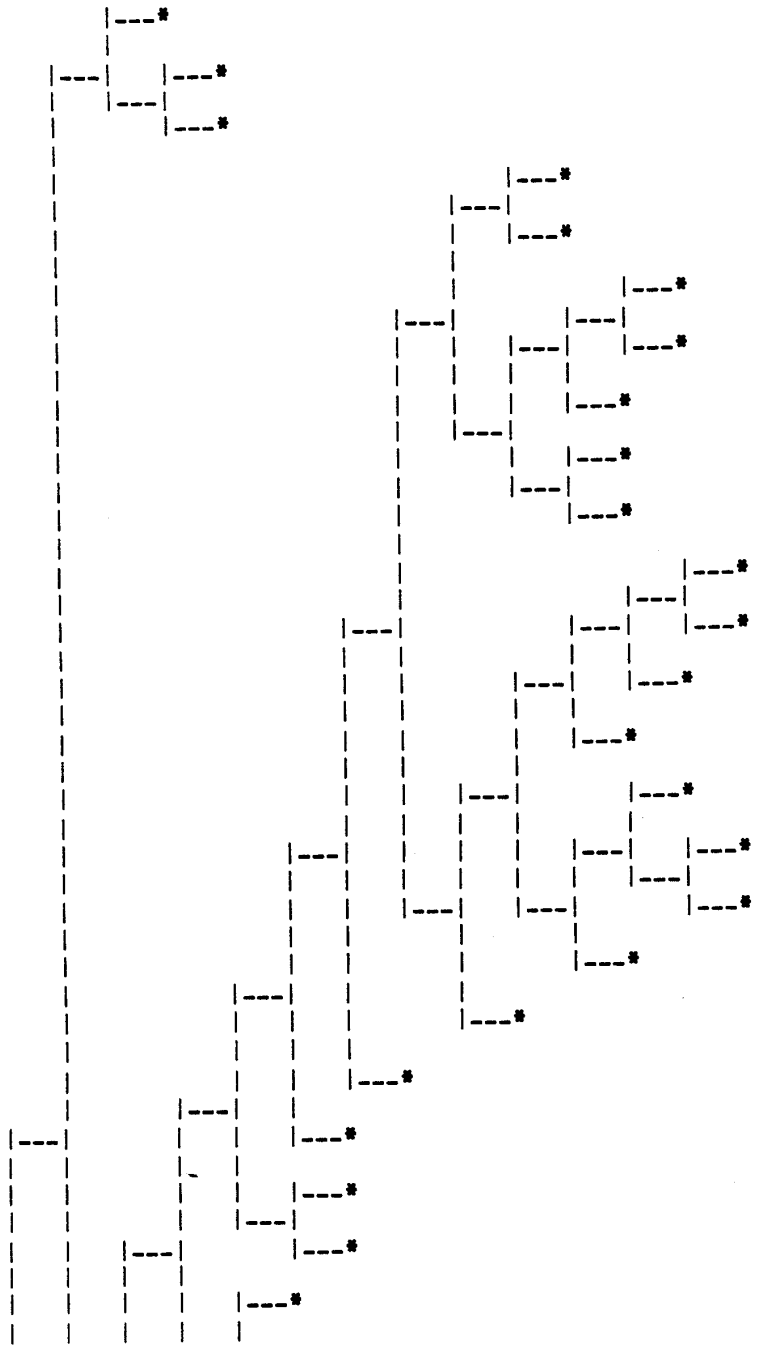
The selector tree representation for the bound DRAWN cover derived by AQ11P has the form:

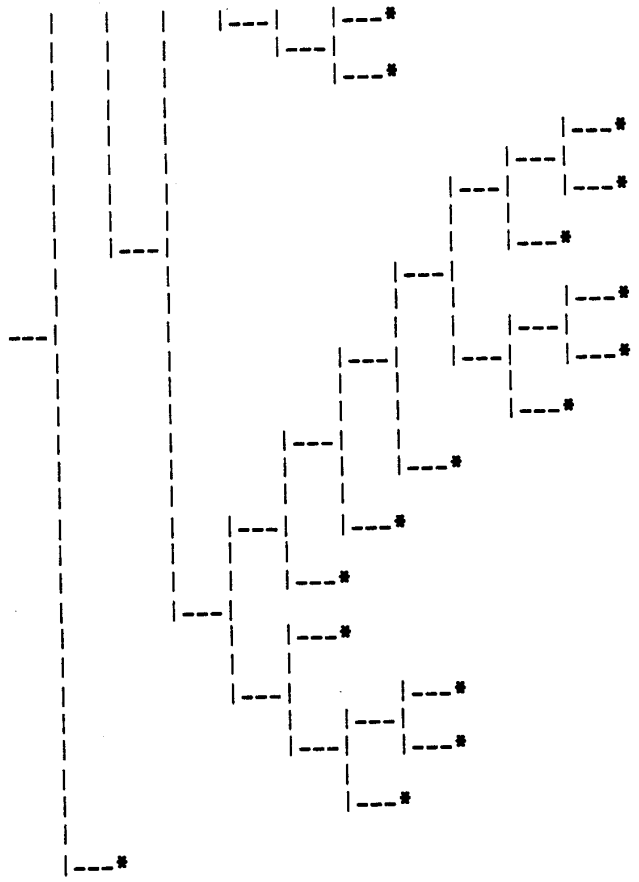


The selector tree for the LOST cover derived by AQ11P can be seen:



For contrast,the selector tree representation of the KPK-decision rule derived by ID-3 has the following form.





6 Conclusions

After detailed review and comparison of methods for learning decision rules from examples developed by R.S.Michalski and J.R. Quinlan, the performance of implementations of these inductive learning systems on a chess pattern learning problem was examined.

With respect to the cost of rule production, we found that AQ11P used from 1 to 2 times the amount of storage used by ID-3 and from about 5 to 100 times the computation time, depending on the incremental rule generation strategy and the number of training instances used. ID-3 generated decision trees for KPK in ~ 10 seconds both with and without incremental rule generation. AQ11P required more than 300 seconds without rule-refinement. The least expensive rule-refinement scheme for AQ11P appeared to be "single expansion", which took 69 seconds for KPK. In every trial, AQ rule-refinement exhibited substantial speedup, but the new covers it produced were always significantly more complicated than the old covers.

With respect to the value or quality of the rules produced, we concentrated on their complexity. This reflects the view that, next to accuracy, the human-comprehensibility of rules intended for rule-based expert systems is most important. Before we could attempt to quantify "incomprehensibility", we had to introduce a general decision structure which subsumed decision trees and covers as special cases. After introducing "selector-trees", we defined a measure on them intended to capture the notion of "cost of understanding." We found, in many examples, that covers are less than half as costly (according to this measure) as decision trees for binary decision problems. We even found this to be true of the best KPK-BTM tree and covers known to us. In sum, the rules produced by AQ11P seem to be much more expensive than those of ID-3, but they are much more comprehensible.

Finally, improvements have been proposed. The most important suggestions show how to make AQ's incremental rule generation more efficient (by eliminating some unnecessary operations) and how to improve the quality of the rules produced (essentially by constructing a sort of discrimination tree which takes advantage of common terms in complexes by binding them together).

7 Bibliography

Brunell,G.A. and Soukup,W.A. (1980?) An Inductive Learning Program, AQ11P. Description, Users Manual, and Examples. Internal Report, Intelligent Systems Group, UIUCDCS.

Bruner,J.S., Goodnow,J.J. and Austin,G.A. (1956) A Study of Thinking

Charniak,E. Riesbeck,C. and McDermott, D. (1980) Artificial Intelligence Programming, Lawrence Erlbaum Associates, Inc.

Feigenbaum,E.A. (1979) Themes and Case Studies of Knowledge Engineering. in Expert Systems in the Micro-Electronic Age (ed. D. Michie) pp.3-25 Edinburgh : Edinburgh University Press.

Hovland,C.I. and Hunt, E.B. (1961) Programming a Model of Human Concept Formation. in Computers and Thought (1963) (eds. Feigenbaum and Feldman),p310- Also in Proc. of the Western Joint Computer Conference (1961),19:145-155.

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

Larson,J.B. (1977) Inductive Inference in the Variable-Valued Predicate Logic System VL21 : Methodology and Computer Implementation. Phd. Thesis, sections 5.3,5.4,5.5,5.6, and Appendix B. Department of Computer Science, University of Illinois at Urbana-Champaign. May 1977.

McEliece, R.J. (1977) The Theory of Information and Coding : A Mathematical Framework for Communication, Addison-Wesley.

Michalski,R.S. (1969) Algorithm AQ for the Quasi-minimal Solution of the Covering Problem, Archiwum Automatyki i Telemechaniki, No. 4, Polish Academy of Sciences, 1969 (in Polish).

Michalski,R.S. (1971) A Geometrical Model for the Synthesis of Interval Covers, Rpt. No. 461, Dept. of Computer Science, Univ. of Illinois, Urbana, Illinois, July 24, 1971.

Michalski,R.S. (1975A) Synthesis of Optimal and Quasi-Optimal Variable-Valued Logic Formulas, Proceedings of the 5th International Symposium on Multiple Valued Logic , Bloomington, Indiana, May 13-16,1975.

Michalski, R.S. (1978) Designing Extended Entry Decision Tables and Optimal Decision Trees Using Decision Diagrams, Dept. of Computer Sci., U. of Illinois at Urbana-Champaign, Rpt. No. UIUCDCS-R-78-898, March 1978.

Michalski,R.S. (1980) Pattern Recognition as Rule Guided Inductive Inference. IEEE Transactions on Pattern Analysis and Machine Intelligence , Vol.PAMI-2,No.4.

Michalski,R.S. and Chilauski,R.L. (1980) Learning by Being Told and Learning from Examples : An Experimental Comparison of the Two Methods of Knowledge Aquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. International Journal of Policy Analysis and Information Systems. Vol.4, No.2, 1980.

Michalski,R.S. and Larson,J.B. (1978) Selection of Most Representative Training Examples and Incremental Generation of VL1 Hypotheses : the Underlying Methodology and the Description of Programs ESEL and AQ11. Report 867, Dept. of Computer Science, University of Illinois (UIUCDCS-R-78-867 Urbana, May 1978).

Michalski,R.S., and McCormick,B.H., (1971) Interval Generalization of Switching Theory, Report No. 442, Dept. of Comp. Sci., Univ. of Illinois, Urbana, May 3, 1971.

Michalski,R.S. and Negri,P. (1977) An Experiment in Inductive Learning in Chess Endgames. Machine Intelligence 8 , Ellis Horwood Ltd. pp.175-192.

Michie,D. (1980B) The State of the Art in Machine Learning. CS397DM Class Note, Fall semester 1980, UIUCDCS.

Niblett,T. and Shapiro,A. (1980) Automatic Induction of Classification Rules for a Chess Endgame. MIP-R-129 Machine Intelligence Research Unit, University of Edinburgh. (September 1980).

Nilsson,N.J. (1980) Principles of Artificial Intelligence, Tioga Publishing Co., Palo Alto, California.

Quinlan,J.R. (1979) Discovering Rules by Induction from Large Collections of Examples. In Expert Systems in the Micro Electronic Age ,(ed. D. Michie) pp.168-201. Edinburgh : Edinburgh University Press.

Quinlan,J.R. (1980) Semi-Autonomous Aquisition of Pattern-Based Knowledge. In Expert Systems (Proceedings of the 69th Infotech State of the Art Conference),pp.11/3-11/15. Maidenhead : Infotech Ltd., London.

Reingold,E.M., Nievergelt,J. and Deo,N. (1977) Combinatorial Algorithms : Theory and Practice , Prentice-Hall, Englewood Cliffs, N.J. USA 07632.

8 Appendix 1 : KPK Features

For completeness, we insert a list of the 31 features (from [NIBLETT and SHAPIRO 80], and personal conversations with Tim Niblett). The feature names in parentheses appear in their report.

1) sint (INTERFERE) : Black can prevent MAINPATT, CANRUN, but not both.

2) srfil (ROOKPAWNRELEVANT) : The pawn is on a rook's file.

3) spran (RANK56RELEVANT) : The pawn's rank is five or six and 'tis not a rookpawn.

4) spra7 (RANK7RELEVANT) : The pawn is on the seventh rank and is not on a rook's file.

5) 5r6p (R6PATT) : Special pattern. See [Niblett and Shapiro 80].

6) 5mp_ (MP56) : By moving the king alone White can get to MAINPATT.

7) 5btop (BTOP1) : The black king is or can get directly in front of the pawn.

8) 5diro (DIROP56) : The 6th rank pattern holds or can be achieved.

9) mpmov (PMOVE) : By first moving the pawn, White can get to MAINPATT.

10) mmp2 (MP2) : Moving the White king alone, White can get to MAINPATT two ranks ahead of the pawn.

11) mmp1 (MP1) : Moving the White king alone, White can get to MAINPATT one rank ahead of the pawn.

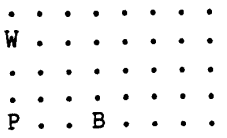
12) many (ANYOP) : White has the distant opposition and can get ahead of the pawn.

13) mdiro (DIROPW) : White's king is one rank ahead of the pawn and has the opposition.

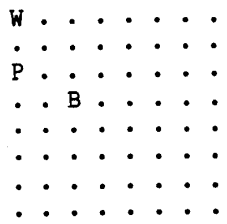
14) mdirb (DIROPB) : Black's king has the opposition and the White king is not more than one rank ahead of the pawn.

15) mnear (NEARERX) : Black's king can get ahead of the White pawn, disallowing MAINPATT.

16) rstal (STALEMATE) : True of some positions which lead to stalemate for rookpawns when White is trapped on the edge of the board. Specifically: the white king must be 3 ranks ahead of the pawn on the same file, and Black's king must be 3 files to the right on the same rank as the pawn. If the pawn is not on the rook's file the position may be lost. An example is:



Another, special case for which rstal is true is:



17) rnear (NEAR2P) : Special pattern.See [NIBLETT and SHAPIRO 80].

18) rrp2 (RP2) : The Black king can trap the White king on the edge of the board.

19) rrp1 (RP1) : Special pattern. See [NIBLETT and SHAPIRO 80].

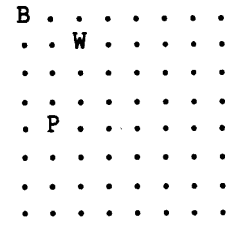
20) rpsq (PSQUARE) : The Black king can move inside the pawn's "square."

21) rneap (NEARP) : The Black king is as near the pawn as White's king.

22) rnea8 (NEARA8) : The Black king can reach A8 before the White king.

23) rneac (NEARC8) : The Black king can reach C8 before the White king.

24) cpat2 (STLMT) : Six positions for which either the Black king is in stalemate, or if White just advances the pawn, stalemate results. Two of these positions are LOST because White may use his king. An example is:



25) cpat1 (PATT1) : Special pattern: Black's king forced to retreat allowing the pawn to run. Specifically: board positions for which the black king is on the first file, the pawn is on the same rank, second

file, the white king is on the third file and either on the same rank or the one just ahead.

26) ccrit (CRIT) : White's king has control over the seventh rank square covering the queening square on the Black king's side of the pawn.

27) cwksa (WKAHD) : The White King is ahead of the pawn, on the same file.

28) cahea (AHEAD) : The Black King can get ahead of the pawn on the pawn's file, before the eighth rank.

29) cplu1 (PLUS1) : The distance of the pawn to the queening square is > the Black king's effective distance minus 1.

30) cplu2 (PLUS2) : The distance of the pawn to the queening square is > the Black king's effective distance minus 2.

31) cimmt (IMCAP) : The Black king can immediately capture the pawn.

BIBLIOGRAPHIC DATA SHEET	1. Report No. ISG 82-2 UIUCDCS-F-82-899	2.	3. Recipient's Accession No.
4. Title and Subtitle A Comparative Study of Inductive Learning Systems AQ11P and ID-3 Using a Chess Endgame Test Problem			5. Report Date September 1982 6.
7. Author(s) Paul O'Rorke			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL			10. Project/Task/Work Unit No. 11. Contract/Grant No. MCS 79-06614 MCS 82-05166
12. Sponsoring Organization Name and Address National Science Foundation Washington, DC			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

Abstract : Inductive learning systems enable computers to learn by generalizing from examples. Such systems may become important tools facilitating construction of expert systems (computerized consultants). This report examines some induction methods of R.S. Michalski and J.R. Quinlan. It also contains an experimental comparison of the performances of the two programs AQ11P and ID-3 on an example problem.

17. Key Words and Document Analysis. 17a. Descriptors

Keywords : artificial intelligence, knowledge engineering, pattern directed inference systems, rule-based expert systems, automated knowledge acquisition, automated induction.

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 67
	20. Security Class (This Page UNCLASSIFIED	22. Price