12

Generating Expert Rules from Examples in PROLOG

B. Arbab*
Department of Computer Science,
University of California at Los Angeles, USA
D. Michie
The Turing Institute,
Glasgow, UK

Abstract

Automatic decision-tree generators have been used in the production of expert systems. A number of experiments indicate the need for more linear (and hence more understandable) yet efficient decision trees. An algorithm is implemented for constructing decision trees optimized with respect to linearity. It also improves on a previous linearizing algorithm (AOCDL) with respect to execution efficiency.

1. PROBLEM DESCRIPTION AND ASSUMPTIONS

This work describes tools for generating decision-trees that are optimized with respect to linearity and are more efficient than those generated by Bratko's AOCDL [1]. The rule generator is specialized so as to obey stated constraints corresponding to the above two properties. Rule induction takes advantage of one of the expert's most reliable and highly developed skills [2], teaching by example, and this avoids the need to resort to dialogue-acquisition of rules, traditionally recognized as the bottle-neck problem of knowledge engineering. However, decision-trees derived from situation-action pairs are inherently less descriptive for expressing concepts than first-order or multi-valued logic used in other projects [3–5]. The lack of descriptive power is primarily associated with the absence of quantified variables.

Quinlan [6] and Shapiro [7] have demonstrated that generation of decision-trees from a set of examples provided by a domain expert is a practical method for knowledge acquisition (see also Martelli and Montanari [8] for generation of optimal trees). This paper stems from two previous approaches to the rule-induction problem: (i) Quinlan's ID3 uses an information theoretic approach to control a 'best-first no

* Present address: IBM Los Angeles Scientific Center, 11601 Wilshire Boulevard, Los Angeles CA 90025-1738.

backtrack' search, producing decision-trees of high, but not optimal, execution efficiency; (ii) Bratko's AOCDL uses backtrack heuristic search. ID3 ignores the human understandability criterion for induced rules while AOCDL ignores the efficiency criterion. ID3's attribute selection criterion, based on entropy, promotes efficient decision-tree execution on the machine. However, the decision-trees are not easily understood by humans. AOCDL is heuristically guided to minimize a non-linearity (branching) measure. Arbitrarily branching structures are hard for a human to keep mental track of. So one idea is to only allow for linear or almost linear decision-trees [9]. A decision-tree is said to be linear if every node has at most one non-terminal son. Note that even trees with high branching ratios (multiple-value attributes) and multiple decision classes can be linear. The relation between linear trees and understandability has been experimentally investigated by Shapiro and Niblett [7, 10]. In two separate classification tasks in chess end-games, structured representations with tree-linearity constraint were uniformly understandable, whereas representations in the form of arbitrarily branching decision trees were uniformly opaque. Our Rule Generator (RG), which was implemented in PROLOG produces decision-trees that are linear where such trees exist. In cases where such a tree does not exist, the most linear tree is constructed. The derived trees are efficient at execution time. These two requirements, linearity and efficiency, are inversely related. A balanced tree is shallower and more efficient for machine execution than a linear tree. In synthesizing decision-trees, however, we always trade efficiency for linearity, in much the same way that structured programming trades efficiency for program clarity and readability.

The presence of a domain expert makes 'structured induction' possible, which breaks the problem into subproblems. A detailed description of structured induction is given by Shapiro and Niblett [7,10]. With structured induction the size of the example sets is never large, e.g. at most in the order of tens. It has been found by Quinlan [6] that small example sets are sufficient to generate rules capable of classifying even large domains with high reliability.

We have developed RG under the assumptions that:

1. Structured induction is feasible.
2. Linearity of decision-trees is to be optimized even at the expense of efficiency.
3. Efficiency of decision-trees is to be increased only subject to the constraint that linearity is *not* affected.

The decision-trees in Figure 1 correspond to an example set taken from a planning domain for building an arch, see [4]. Each node of the tree corresponds to an attribute, leaf nodes represent decision classes, and the labels on the arcs are the attribute's values. These trees were induced by: (1) Expert-Ease [12], a commercial version of the ID3-derived ACLS

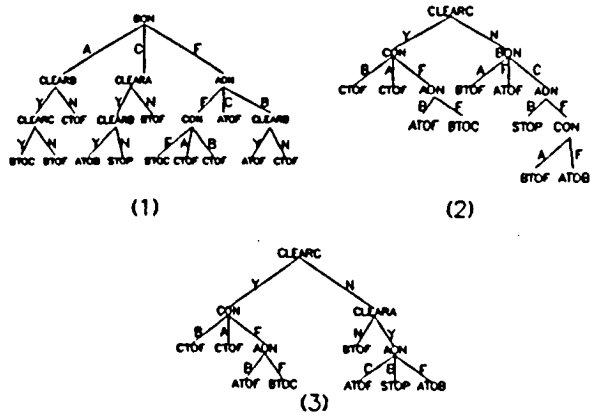


Figure 1. Trees with different linearity and efficiency.

algorithm which produces efficient but non-linear trees; (2) AOCDL, which maximizes linearity but not efficiency; and (3) RG, which maximizes linearity and promotes efficiency. The exact non-linearity and efficiency (execution cost) measures of these trees can be seen for comparison in Figure 10 under EX4.

Tree (1) has eight non-terminals (attributes) and 12 terminal (class) nodes, while tree (3) has five attributes and eight classes. Tree (1) has an irregular branching structure, while tree (3) has a small non-linearity that occurs at the root. Decision trees can be used to classify new examples. An interpreter for executing such trees will compute a value for an attribute (non-terminal). This value specifies which branch of the tree should be traversed next. A classification has been made when a class (terminal) node is reached.

RG incorporates linearity and efficiency measures within an AO* [13] algorithm as heuristics to guarantee optimal linearity. The efficiency of the decision-trees is increased according to each candidate attribute's expected information contribution if appended at the given point in the tree, i.e. attributes with high information content will be placed as high in the decision tree as possible thus increasing the probability of classification to occur as early as possible.

The following points should be read against the background of examples such as those worked in the Appendix. They concern theoretical limitations of the rule synthesis process and how this process may be used for solving problems. These three properties of induction define, more precisely, what can be expected from the rule synthesis process.

1. Any rule synthesized from only a subset of all possible examples, cannot in general be proved correct in all cases (i.e. all possible examples). However, this does not imply that correct rules for all possible cases cannot be constructed from a minimal set of examples.

2. In applications where correct classification is required of finite sets of examples taken from a finite domain, one can consider a table look-up process. It may still be desirable to use rule synthesis in such cases since large tabulations can be condensed, without loss of information or accuracy, into simple rules.

3. The synthesis process cannot by itself find any hidden hierarchical structures in example sets. In structured induction, these structures are thought out by the expert and documented before the rule induction process is started. A structure set by the expert can later be modified as the project progresses.

The generated rules are semantically equivalent to the example sets they originated from when these are complete, i.e. no information has been added or lost by the rule-generating program. It is this characteristic of rule-generating programs that sets them apart from generalization techniques used in other projects such as Marvin [14] or MIS [15].

2. DECIDER STATUS OF ATTRIBUTES

Experts are generally adept at communicating their expertise by means of examples. The examples thus form a language through which knowledge is communicated. There are three parts to this language: attributes, classes, and examples, the latter being defined in terms of attribute values and classes. Attributes are defined by the domain specialist as critical features or relevant facts, e.g. colour of eyes with values brown, black, and blue; or kidney size with values large, small, and normal. Class values correspond to the identification made on the basis of attribute values, e.g. normal, or kidney stone. Examples are represented as attribute-value vectors paired with classes as shown in Figure 2. One additional criterion imposed on examples is that they must be clash free. A clash is defined as two identical vectors leading to different class values. Clashes usually signify the inadequacy of attributes

A1	A2	A3	A4	A5	class
t	t	f	t	t	c1
t	f	t	f	f	c1
t	f	f	t	f	c1
f	t	f	t	t	c2
t	t	f	t	f	c2
t	f	t	t	f	c2
f	f	t	f	t	c1
t	f	f	t	t	c1
t	t	t	t	f	c2
f	f	f	f	f	c1
f	t	f	f	t	c2

Figure 2. An example set.

for classification of the problem and can be removed by introducing additional attributes.

With respect to a specified example set, an attribute has a decider status: total, partial, or non-decider. 'Decisiveness' of an attribute may be computed from a matrix whose rows correspond to class values and columns to values the attribute of interest may range over. Each entry in this matrix corresponds to a frequency count of the class per attribute's value. An attribute's decider status is defined as follows:

1. Total Decider, if the attribute partitions the example set such that each partition belongs to a single class. In matrix form this corresponds to the condition that there is at most one non-zero value in each column.
2. Partial Decider, if the attribute partitions the example set such that all but one partition belong to a single class. In matrix form this corresponds to at least one column where all entries except one are zero.
3. Non-Decider, if neither of the above is true.

In Figure 3 we show an example set and the matrices corresponding to each attribute's decisiveness status. Attribute $A1$ is a total decider, $A2$ a partial decider and $A3$ a non-decider.

Decider status of an attribute plays an important role in our search for linear decision-trees. If an attribute must be selected from a set of total or partial deciders, then the linearity of the final tree is not affected by the choice of a particular attribute, but efficiency can depend on this choice. However, selection of a non-decider attribute can affect efficiency and invariably destroys linearity. Consider the example set in Figure 2 where for simplicity we have assumed binary attributes and only two classes.

It so happens that in the above example set all candidate attributes for the top of the tree are non-deciders. However, different attributes lead to various non-linear trees. Using attribute $A2$ at the top leads to a tree of the form shown on the left side of Figure 4 while using any of attributes $A1$, $A3$, $A4$, or $A5$ leads to a tree of the form shown on the right.

Clearly the tree on the right is a more linear tree (we recall that linear decision-trees are easier to understand). Thus, when selecting an attribute from a set of non-deciders one must consider their effect on the

A1	A2	A3	Class
t	t	t	c1
t	t	f	c1
f	f	f	c2
f	t	t	c2

A1	f	t
c1	0	2
c2	2	0

A2	t	f
c1	2	0
c2	1	1

A3	t	f
c1	1	1
c2	1	1

Figure 3. computation of decider status.

Figure 4. Trees with different linearity measures.

overall linearity and efficiency of the decision-tree. In general, selecting the right attribute requires a search procedure which is described in later sections.

3. LINEARITY MEASURE

Degree of linearity has been used as a measure of desirability for trees. This concept must be formalized to allow comparison of trees on the basis of their non-linearity. Some desirable characteristics of a function to compute non-linearity of trees are:

1. An intuitive, yet formal, basis.
2. Sensitivity to the size of trees.
3. Sensitivity to location of non-linearity in a tree.

Bratko [1] has proposed such a function. The non-linearity measure which he proposes is based on the fact that traversal of a linear tree requires scanning through contiguous memory locations and minimizes jumps across the memory. This may be one reason why linear decision-trees are easier for humans to understand than are non-linear trees. Let T be a decision tree whose root is A and subtrees are $S1, S2, \ldots, Sm$, as in Figure 5.

The proposed measure for non-linearity is:

$$NL(T) = (1/m) \times \sum_{i=1}^{m} [NL(Si) + (m - i) \times s(Si)]$$

where $NL(T)$ denotes the non-linearity of T, $NL(T) = 0$ when T is a class value (leaf node) and the number of internal nodes of the tree, $s(T)$ is

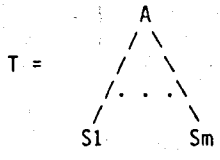
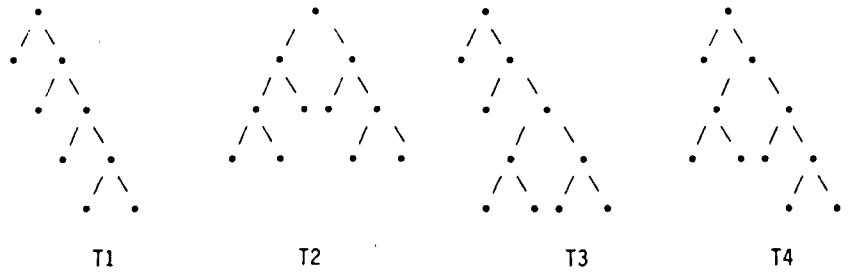


Figure 5. An abstract tree.



NL(T1) = 0
NL(T2) = (1 / 2) × [(2 − 1) × s(S1) + (2 − 2) × s(S2) + NL(S2)] = 1
NL(T3) = 1 / 8
NL(T4) = 1 / 4

Figure 6. Non-linearity measure for four trees.

defined as follows:

$$s(T) = 1 + \sum_{i=1}^{m} s(Si)$$

where $s(T) = 0$ if T is a class value. It is assumed that Si are sorted in increasing order of $s(Si)$. Non-linearities of four trees are shown in Figure 6.

$T1$ is absolutely linear; thus its non-linearity measure is zero. $T2$ is very close to being a balanced tree: non-linearity one. $T3$ is preferred to $T4$, i.e. this function is sensitive to the location of non-linearity within a tree (the lower a non-linearity occurs in a tree the lower (better) its measure).

4. EFFICIENCY MEASURE

Consider the example set of Figure 2. Two equally linear decision-trees for classifying this example set are shown in Figure 7. Labels on the arcs correspond to the number of examples per value of each attribute, Figure

Figure 7. Trees with different execution cost.

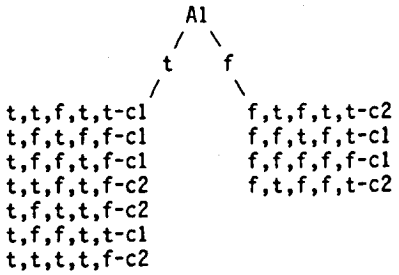


Figure 8. Attribute A1 divides the example set.

8 shows that attribute $A1$ divides the original example set into two example sets of size 7 and 4 each. Let $c(a_i)$ represent the execution cost of an attribute. There are 11 examples in the original example set and the execution cost for each tree can be computed on the basis of how early in the decision tree a classification takes place. One way of computing this cost is as follows:

Average cost for $T1 =$

$$[11 \times c(a_1) + 4 \times c(a_2) + 7 \times c(a_4) + 6 \times c(a_3) + 3 \times c(a_2) + 2 \times c(a_5)]/11$$

Average cost for $T2 =$

$$[11 \times c(a_4) + 4 \times c(a_2) + 7 \times c(a_3) + 5 \times c(a_2) + 3 \times c(a_1) + 2 \times c(a_5)]/11.$$

Note that the difference between $T1$ and $T2$ occurs at the second from the top. Using $T1$ the chance of an example being classified is only 1 out of 7 while using $T2$ it is 2 out of 7. Assuming the execution cost of each attribute has unit cost, $c(a_i) = 1$, the execution cost for trees $T1$ and $T2$ are 3.0 and 2.9 respectively. That is, $T2$ is about 3% more efficient than $T1$. The difference can be larger, see Figure 10 for more examples. Thus, attribute selection can have an effect on the average execution cost of a decision-tree. Clearly, it is desirable for attributes with high information content (entropy) to appear as early as possible in a decision-tree. This increases the probability that a classification will occur as soon as possible. Thus RG employs entropy as the selection criterion for increasing efficiency. The selection criterion must then consider the entropy. The entropy or information content of an attribute can be computed from the following formula [6]. An attribute's entropy is given by $M(C)$ minus $B(C, A)$ where C is the example set and

$$M(C) = \sum_{i=1}^{n} - P_i \times \log_2 P_i$$

$B(C, A) =$ (probability that value for A is A_i) $\times M(C_i)$

where n is the number of classes, P_i is the occurrence probability of the

ith class, A is an attribute, A_i is a value for the attribute A and C_i is the example set after it has been split by A.

Note that the probabilities can be estimated from the relative frequencies in C_i given that C is a representative of the universe. For example, in order to compute the entropy of attribute $A1$ in Figure 2 we first must compute $B(C, A1)$. This computation requires the example set to be divided according to the values of $A1$ as in Figure 8.

The information content of the true and false branches of $A1$ can be computed as follows:

$$M(C_1) = -\tfrac{4}{7} + \log_2 \tfrac{4}{7} - \tfrac{3}{7} \times \log_2 \tfrac{3}{7} = 0.98522$$

$$M(C_2) = -\tfrac{2}{4} \times \log_2 \tfrac{2}{4} - \tfrac{2}{4} \times \log_2 \tfrac{2}{4} = 1.0.$$

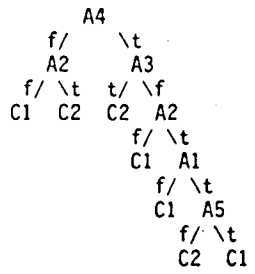
The expected information content, $B(C, A1)$ of $A1$ is:

$$B(C, A1) = \tfrac{7}{11} \times 0{\cdot}98522 + \tfrac{4}{11} \times 1.0 = 0.99059.$$

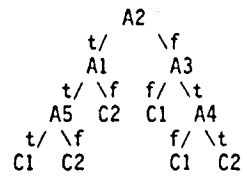
The information content of $A1$ is $M(C)$, where C is the original example set, minus $B(C, A1)$ or $0.99395 - 0.99059 = 0.0036$. The entropy for other attributes can be computed in this fashion. A selection based on this criterion will minimize the expected execution cost of decision-trees.

Entropy has been used without the linearity measure in ID3. It should not be surprising that decision trees constructed in such a fashion tend to be balanced rather than linear. For example, the ID3 solution for the example set in Figure 2 is shown in Figure 9, which also shows the tree generated by our algorithm.

Clearly, ID3 produces more efficient decision trees than RG. However, since we choose to sacrifice efficiency for linearity, decision-trees produced by RG are more desirable. If the added tree-synthesis cost can be accepted, then optimal efficiency (among equally linear candidate trees) can be guaranteed by substituting 'best-first with backtrack' for the 'best-first no backtrack' strategy borrowed from ID3.



RG



As solved by ID3

y measures.

5. OUTLINE OF RG

We adapted Bratko's measure of non-linearity and used an attribute selection criterion that promotes execution efficiency of the resulting decision-tree. RG incorporates the notions of linearity and efficiency into an AO* [13] search technique.

The state space for finding a decision tree is finite and decreasing with the number of variables since the number of attributes and examples are finite. An 'And/Or' tree is used to represent the state space. 'Or' nodes correspond to candidate attributes and 'And' nodes are subproblems that must be solved. The root can be considered as an 'And' node. Each node may be labelled as solved, closed or open. Solved nodes mean that a solution has been reached from this node. A closed node means that a solution under current consideration incorporates this node internally. A node is open if it is neither closed nor solved.

During the expansion of the search tree, an optimistic estimate for non-linearity is used in conformity with the AO* algorithm for searching 'And/Or' graphs. This estimate differentiates between total, partial and non-decider attributes. Thus, if there are total deciders among the candidate attributes, the search tree is expanded using them and the nodes are labelled as solved. All partial decider attributes are considered if no total deciders exist, and non-decider attributes are considered only if there are no total or partial deciders.

The optimal solution path is marked in the search tree according to: (i) non-linearity of the partially constructed decision tree; (ii) number of expected internal modes; (iii) the attribute's entropy measure. The entropy measure is used simply as a tie breaker between attributes which produce equally linear decision-trees. Thus, optimality with respect to linearity is guaranteed while efficiency is only enhanced. When RG terminates, the optimal decision-tree can be constructed by tracing markers from the root node to the bottom and recording the attributes and their values.

6. RESULTS WITH RG

RG was used to induce rules for some examples selected from the planning domain (construction of an arch and sorting a stack of blocks) and chess end-games (some examples from Shapiro's Ph.D. thesis [7]) in addition to some artificially constructed example sets. For the most part the rules synthesized by RG were more linear than those induced by ID3. The exceptions occurred when ID3 happened to construct a fully linear decision-tree. ID3 produces more efficient decision-trees than AOCDL or RG. This is to be expected because RG (and AOCDL) emphasizes the linearity criterion before efficiency. However, the decision-trees generated by RG were more efficient than those produced by AOCDL since this

	RG		AOCDL		ID3	
	NL	Cost	NL	Cost	NL	Cost
EX1	0	2.7	0	2.7	0	2.7
EX2	0.5	2.91	0.5	3.0	1.0	2.54
EX3	0	2.25	0	2.25	0.5	2.0
EX4	1.0	2.76	1.0	3.5	2.11	1.12
EX5	0	3.0	0	3.42	0	3.0

Figure 10. Performance analysis of programs.

```
problem(ex1).

/* Attributes */
att(a1, (t.f.nil)).
att(a2, (f.t.nil)).
att(a3, (f.t.nil)).
att(a4, (f.t.nil)).
att(a5, (f.t.nil)).

/* Classes */
class(c1).
class(c2).

/* Examples */
ex(1,  (t.t.f.t.t.nil), c1).
ex(2,  (t.f.t.f.f.nil), c1).
ex(3,  (t.f.f.t.f.nil), c1).
ex(4,  (f.t.f.t.t.nil), c2).
ex(5,  (t.t.f.t.f.nil), c2).
ex(6,  (t.f.t.t.f.nil), c2).
ex(7,  (f.f.t.f.t.nil), c1).
ex(8,  (t.f.f.t.t.nil), c1).
ex(9,  (t.t.t.t.f.nil), c2).
ex(10, (f.f.f.f.f.nil), c1).
```

Figure 11. Input file for EX1; this can be explained by a linear decision-tree.

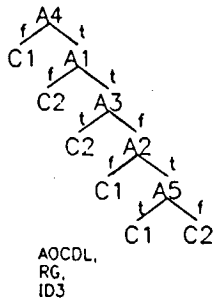


Figure 12. Synthesized decision-trees for EX1.

```
problem(ex2).
/* Attributes */
att(a1, (t.f.nil)).
att(a2, (f.t.nil)).
att(a3, (f.t.nil)).
att(a4, (f.t.nil)).
att(a5, (f.t.nil)).
/* Classes */
class(c1).
class(c2).
/* Examples */
ex(1,  (t.t.f.t.t.nil), c1).
ex(2,  (t.f.t.f.f.nil), c1).
ex(3,  (t.f.f.t.f.nil), c1).
ex(4,  (f.t.f.t.t.nil), c2).
ex(5,  (t.t.f.t.f.nil), c2).
ex(6,  (t.f.t.t.f.nil), c2).
ex(7,  (f.f.t.f.t.nil), c1).
ex(8,  (t.f.f.t.t.nil), c1).
ex(9,  (t.t.t.t.f.nil), c2).
ex(10, (f.f.f.f.f.nil), c1).
ex(11, (f.t.f.f.t.nil), c2).
```

Figure 13. Input file for EX2; this cannot be explained by a linear decision-tree.

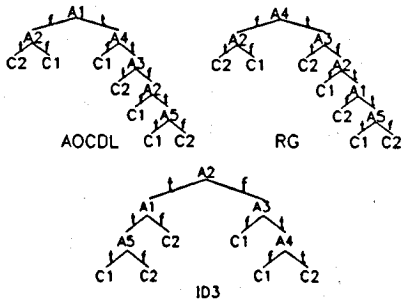


Figure 14. Synthesized decision-tree for EX2.

```
problem(ex3).

/* Attributes */

att(a1, (t.f.nil)).
att(a2, (f.t.nil)).
att(a3, (f.t.nil)).

/* Classes */

class(c1).
class(c2).
class(c3).
class(c4).

/* Examples */

ex(1,  (t.t.f.nil), c1).
ex(2,  (f.t.t.nil), c2).
ex(3,  (f.f.t.nil), c3).
ex(4,  (f.f.f.nil), c4).
```

Figure 15. Input file for EX3; a linear decision-tree but ID3 cannot find it.

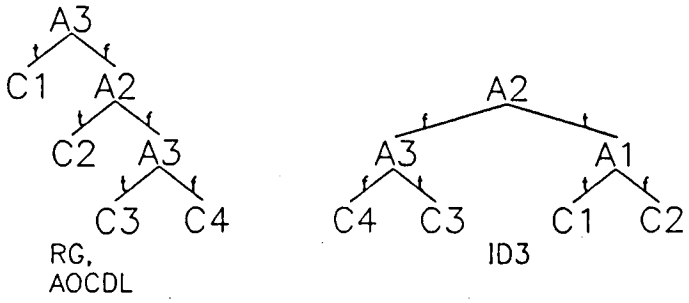


Figure 16. Synthesized decision-trees for EX3.

program, AOCDL, only optimizes the linearity criterion. The decision-trees generated by RG are more understandable than those generated by ID3, since RG promotes efficiency without destroying linearity. For an example see Figure 1. Five examples that demonstrate the differences between these programs are given in Figure 10. Complete listings of the examples may be obtained from the authors. Outline listings of the examples,

```
problem(building_tower).

/* Attributes */
att(aon, (f.b.c.nil)).
att(bon, (f.a.c.nil)).
att(con, (f.a.b.nil)).
att(cleara, (y.n.nil)).
att(clearb, (y.n.nil)).
att(clearc, (y.n.nil)).

/* Classes */

class(stop).
class(btoc).
class(ctof).
class(atob).
class(btof).
class(atof).

/* Examples */

ex(1 ,  (f.f.f.y.y.y.nil), btoc).
ex(2 ,  (f.f.a.n.y.y.nil), ctof).
ex(3 ,  (f.f.b.y.n.y.nil), ctof).
ex(4 ,  (f.a.f.n.y.y.nil), btoc).
ex(5 ,  (f.a.b.n.n.y.nil), ctof).
ex(6 ,  (f.c.f.y.y.n.nil), atob).
ex(7 ,  (b.f.f.y.n.y.nil), atof).
ex(8 ,  (b.f.a.n.n.y.nil), ctof).
ex(9,   (b.c.f.y.n.n.nil), stop).
ex(10,  (f.c.a.n.y.n.nil), btof).
ex(11,  (c.f.f.y.y.n.nil), atof).
ex(12,  (c.f.b.y.n.n.nil), atof).
ex(13,  (c.a.f.n.y.n.nil), btof).
```

Figure 17. Input file for EX4; building a tower in the blocks world.

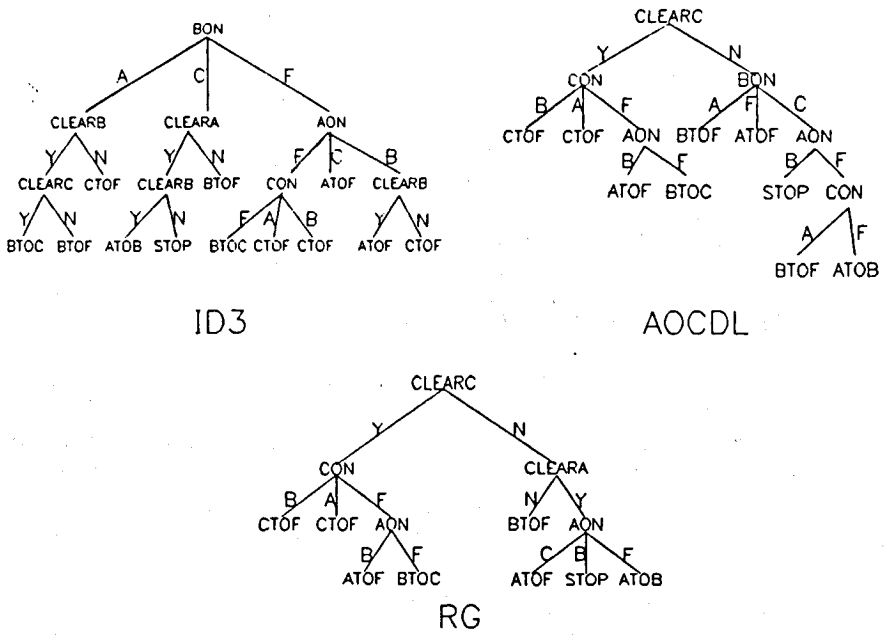


Figure 18. Synthesized decision-trees for EX4.

together with the trees generated by the different algorithms, are shown in Figures 11–20.

The table of Figure 10 and the more detailed listings of Figures 11–20 indicate that RG produces decision-trees that are as linear as those produced by AOCDL but more efficient. Also, decision-trees produced are more linear than those produced by ID3. Thus, RG has been successful in

```
problem(btoqs).

class(true).
class(false).

att(skrxp, f.t.nil). /*Can the BR achieve a skewer or BK attack the WP*/
att(bkona, f.t.nil). /*Is the BK on rank A in a position to aid the BR*/
att(bkon8, f.t.nil). /*Is the BK on file 8 in a position to aid the BR*/
att(bknwy, f.t.nil). /* Is the BK in the BR's way */
att(wkovl, f.t.nil). /* Is the WK overloaded */

ex(1, t.f.f.f.f.nil, true).
ex(2, f.t.f.f.f.nil, true).
ex(3, f.f.t.f.f.nil, true).
ex(4, f.f.f.t.f.nil, false).
ex(5, f.f.f.f.t.nil, true).
ex(6, f.f.f.f.f.nil, false).
ex(7, f.f.f.t.t.nil, false).
```

Figure 19. Input file for EX5; an example from ref. [7].

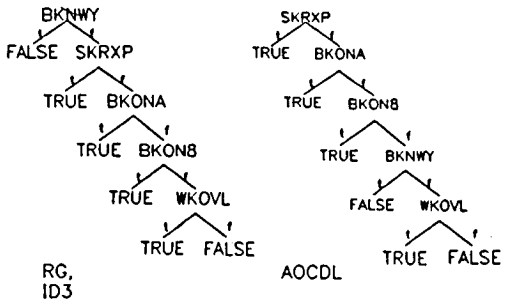


Figure 20. Synthesized decision-trees for EX5.

its goal, i.e. producing the most linear decision-tree while enhancing execution efficiency.

7. SUMMARY

An algorithm, RG, for producing human-understandable yet efficient decision-trees has been described and implemented. RG was implemented in PROLOG and tested using sample problems from Bratko [1] and Shapiro [7]. In all cases of difference, the decision-trees produced were more linear than decision-trees synthesized by ID3 and more efficient that decision-trees generated by AOCDL.

This algorithm is heuristically guided by linearity and efficiency measures resulting in the generation of more understandable decision trees. RG, combined with structured induction promises rapid construction of expert systems by permitting the expert to communicate his knowledge and experience through a simple yet flexible language.

Acknowledgements

We would like to thank Rina Dechter, Jim Moore, and Gary Silverman for providing many helpful suggestions throughout the project. Thanks are also due to the IBM Los Angeles Scientific Center for providing resources and support.

REFERENCES

1. Bratko, I. (1983) Generating human-understandable decision rules, Working paper, E. Kardelj University, Liubljana, Yugoslavia.
2. Michie, D. (1982) The state of the art in machine learning. In *Introductory readings in expert systems* (ed. D. Michie) pp. 208–228. Gordon & Breach, London.
3. Vere, S. A. (1978) Inductive learning of relational productions. In *Pattern-directed inference systems* (eds D. A. Waterman and F. Hayes-Roth) pp. 281–95. Academic Press, New York.
4. Michalski, R. S. (1980) Pattern recognition as rule-guided inductive inference, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-2,** 349–361.
5. Hayes-Roth, F. and McDermott, J. (1977) Knowledge acquisition from structural description. *Proc. IJCAI-5,* pp. 356–362.

6. Quinlan, J. R. (1983) Learning efficient classification procedures and their applications to chess end-games In *Machine learning: an artificial intelligence approach* (eds R. S. Michalski, J. Carbonell, and T. Mitchell) pp. 463–82. Tioga, Palo Alto, Calif.
7. Shapiro, A. (1987) *Structured induction in expert systems.* Addison Wesley, Wokingham, England, and New York.
8. Martelli, A. and Montanari, U. (1973) Optimizing decision trees through heuristically guided search, *Commun. ACM* **21,** 1025.
9. Michie, D. (1981) 'Mind-like' capabilities in computers: a note on computer induction. *Cognition* **12,** 97–108.
10. Shapiro, A. and Niblett, T. (1982) Automatic induction of classification rules for a chess endgame. In *Advances in computer chess 3* (ed M. R. B. Clarke) pp. 73–92. Pergamon Press, Oxford.
11. Dechter, R. and Michie, D. (1984) Structured induction of plans and programs, IBM Los Angeles Scientific Center report.
12. McLaren, R. (1983) *Expert-Ease user manual.* Intelligent Terminals Ltd., Glasgow.
13. Nilsson, N. J. (1980) *Principles of artificial intelligence.* Tioga, Palo Alto, Calif.
14. Sammut, C. and Banerji, R. B. (1981) Learning concepts by asking questions, Working paper, Department of Computer Science, University of Illinois at Urbana-Champaign.
15. Shapiro, E. Y. (1981) *Inductive inference of theories from facts.* Department of Computer Science, Yale University.

FURTHER READING

Arbab, B. and Michie, D. (1985) Generating rules from examples. *Proc. IJCAI-9.*
Attneave, F. (1959) Applications of information theory to psychology: A summary of basic concepts, methods and results. University of Oregon.
Paterson, A. and Niblett, T. (1982) *ACLS user manual.* Intelligent Terminals Ltd., Edinburgh.
Quinlan, J. R. (1982) Semi-autonomous acquisition of pattern-based knowledge. In *Machine Intelligence 10* (eds J. E. Hayes, D. Michie, and Y.-H. Pao). Ellis Horwood, Chichester.
Roberts, G. M. (1977) M.S. thesis, An implementation of PROLOG. University of Waterloo, Department of Computer Science.
Roberts, G. M. (1983) PROLOG *user's manual, version 1.4.* University of Waterloo, Department of Computer Science.