P93 30

Journal of Intelligent Information Systems, 2, 279–304 (1993)
© 1993 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Learning Decision Trees from Decision Rules: A Method and Initial Results from a Comparative Study

I.F. IMAM AND R.S. MICHALSKI iimam@aic.gmu.edu, michalski@aic.gmu.edu
Center for Artificial Intelligence, George Mason University, Fairfax, VA 22030.

Abstract. A standard approach to determining decision trees is to learn them from examples. A disadvantage of this approach is that once a decision tree is learned, it is difficult to modify it to suit different decision making situations. Such problems arise, for example, when an attribute assigned to some node cannot be measured, or there is a significant change in the costs of measuring attributes or in the frequency distribution of events from different decision classes. An attractive approach to resolving this problem is to learn and store knowledge in the form of decision rules, and to generate from them, whenever needed, a decision tree that is most suitable in a given situation. An additional advantage of such an approach is that it facilitates building *compact decision trees*, which can be much simpler than the logically equivalent conventional decision trees (by compact trees are meant decision trees that may contain branches assigned a *set of values*, and nodes assigned *derived* attributes, i.e., attributes that are logical or mathematical functions of the original ones). The paper describes an efficient method, AQDT-1, that takes decision rules generated by an AQ-type learning system (AQ15 or AQ17), and builds from them a decision tree optimizing a given optimality criterion. The method can work in two modes: the *standard mode*, which produces conventional decision trees, and *compact mode*, which produces compact decision trees. The preliminary experiments with AQDT-1 have shown that the decision trees generated by it from decision rules (conventional and compact) have outperformed those generated from examples by the well-known C4.5 program both in terms of their simplicity and their predictive accuracy.

Keywords: machine learning, inductive learning, decision trees, decision rules, attribute selection

1. Introduction

Methods for learning decision trees from examples have been very popular in machine learning due to their simplicity. Decision trees built this way can be quite efficient, as long as they are used in decision making situations for which they are optimized and these situations remain relatively stable. Problems arise when these situations significantly change and the assumptions under which the tree was built do not hold anymore. For example, in some situations it may be difficult to determine the value of the attribute assigned to some node. One would like to avoid measuring this attribute and still be able to classify the example, if this is potentially possible (Quinlan, 1990). If the cost of measuring various attributes changes, it is desirable to restructure the tree so that the "inexpensive" attributes are evaluated first. A tree restructuring is also desirable, if there is a significant change in the frequency of occurrence of examples from different classes. A restructuring of a decision tree to suit the above requirements is, however, difficult to do. The reason for this is that decision trees are a form of procedural knowledge representation, and imposes constraints on the evaluation order of the attributes that are not logically necessary.

An attractive alternative that avoids such problems is to learn and store knowledge in the form of decision rules, and to generate from them an appropriate decision tree "dynamically," as needed. Decision rules represent knowledge declaratively, and thus do not impose any order on the evaluation of the attributes. Due to the lack of "order constraints," rules can be evaluated in many different ways. Since the number of rules per class is typically much smaller than the number of examples per class, generating a decision tree from rules can potentially be done "on line," without a delay noticeable to the user. Such "virtual" decision trees can be tailored to any given decision making situation.

This approach may allow one, for example, to generate a decision tree that avoids evaluating an attribute that is difficult or impossible to measure for some decision making situation, or that fits well a particular frequency distribution of decision classes. In some situations, it may be unnecessary to generate a complete decision tree, but instead to generate only the part of it, whose leaves are associated only with decision classes of interest. Thus, such an approach has many potential advantages.

On the other hand, it has a disadvantage that it requires determining decision rules first. There exist, however, efficient methods for generating decision rules from examples (rules can also be generated by an expert). The needed decision rules have to be generated only once, and then they can be used many times for generating decision trees according to changing requirements of decision making situations.

This paper presents a simple and efficient method for generating decision trees from decision rules. It also reports preliminary results from experiments comparing it with a well-known C4.5 method for learning decision trees from examples.

2. A brief description of the AQ15 rule learning program

The proposed method, called AQDT-1 (AQ-derived Decision Tree - 1), generates decision trees from decision rules. The decision rules used by the program are generated from examples by an AQ-type inductive learning system, specifically, by AQ15 (Michalski et al., 1986) or AQ17-DCI (Bloedorn and Michalski, 1991a, b). In order to make the paper self-contained, we briefly describe the AQ15 and AQ17-DCI.

AQ15 learns decision rules for a given set of decision classes from examples of decisions, using the STAR methodology (Michalski, 1983). The simplest algorithm based on this methodology, called AQ, starts with a "seed" example of a given decision class, and generates a set of the most general conjunctive descriptions of the seed (alternative decision rules for the seed example). Such a set is called the "star" of the seed example. The algorithm selects from the star a description that optimizes a criterion reflecting the needs of the problem domain. If the criterion is not defined, the program uses a default criterion that selects the description that covers the largest number of positive examples (to minimize the total number of rules needed) and, with the second priority, that involves the smallest number of attributes (to minimize the number of attributes needed for arriving at a decision).

If the selected description does not cover all examples of a given decision class, a new seed is selected from uncovered examples, and the process continues until a complete class description is generated. The algorithm can work with few examples or with many examples, and optimize the description according to a variety of easily-modifiable hypothesis quality criteria.

The learned descriptions are represented in the form of a set of decision rules, expressed in an attributional logic calculus, called *variable-valued logic 1* or *VL1* (Michalski, 1973). A distinctive feature of this representation is that it employs, in addition to standard logic operators, the internal disjunction operator (a disjunction of values of the same attribute in a condition) and the "range" operator (to express conditions involving a range of discrete or continuous values). These operators help to simplify rules involving multivalued discrete attributes; the second operator is also used for creating logical expressions involving continuous attributes.

AQ15 can generate decision rules that represent either *characteristic* or *discriminant* concept descriptions, depending on the settings of its parameters (Michalski, 1983). A characteristic description states properties that are true for all objects in the concept. The simplest characteristic concept description is in the form of a single conjunctive rule (in general, it can be a set of such rules). The most desirable is the maximal characteristic description, that is a rule with the *longest* condition part, i.e., stating as many common properties of objects of the given class as can be determined. A discriminant description states properties that discriminate a given concept from a fixed set of other concepts. The most desirable is the minimal discriminant descriptions, that is a rule with the *shortest* condition part. For example, to distinguish a given set of tables from a set of chairs, one may only need to indicate that tables "have large flat top." A characteristic description of the tables would include also properties such as "have four legs, have no back, have four corners, etc." Discriminant descriptions are usually much shorter than characteristic descriptions.

Another option provided in AQ15 controls the relationship among the generated descriptions ("rulesets" or "covers") of different decision classes. In the "IC" (Intersecting Covers) mode, rulesets of different classes may logically intersect over areas of the description space in which there are no training examples. In the "DC" (Disjoint Covers) mode, descriptions of different classes are logically disjoint. The DC mode descriptions are usually more complex, both in the number of rules and the number of conditions. There is also a "DL" mode (a Decision List mode, also called "VL" mode—for variable-valued logic mode), in which the program generates rulesets that are linearly ordered. To assign a decision to an example using such rulesets, the program evaluates them in order. If ruleset *i* is satisfied by the example, then the decision is made, otherwise, the program proceeds to the evaluation of the ruleset *i+1*. In IC and DC modes, rulesets can be evaluated in any order.

Alternatively, the system can use rules from the AQ17-DCI program for Data-driven Constructive Induction. AQ17-DCI differs from AQ15 mainly in that it contains a module for generating additional attributes. These attributes are various logical or mathematical combinations the original attributes. The program generates a large number of potential new attributes, and selects from them those most promising based on an "attribute quality" criterion.

To illustrate the format of rules generated by AQ15 (or AQ17-DCI), an exemplary ruleset is shown in Figure 1. The ruleset (that can be re-represented as a disjunctive normal form expression) describes a voting record of Democratic Representatives in the US Congress.

```
R1:  [Gas_cont_ban = yes] & [Soc_sec_cut = no v not-reg.]
R2:  [Draft = Yes v vote not-reg.] & [Alaska_parks = yes v vote
                not registered.] & [Food_stamp_cap = no] &
                [State = northeast v northwest]
R3:  [Chrysler = yes v not-reg.] & [Income = low]
R4:  [Education = yes] & [Occupation = yes]
```

Figure 1. A ruleset generated by AQ15 for the concept "Voting Pattern of Democratic Representatives."

Each rule is a conjunction of elementary conditions. Each condition expresses a simple relational statement. For example, the condition [State = northeast v northwest] states that the attribute "State" (of the Representative) should take the value 'northeast' or 'northwest' to satisfy the condition.

The above rules were generated from examples of the voting records. For illustration, below is an example of a voting record by a Democratic representative:

Draft registration=no; Ban of aid to Nicaragua=no; Cut expenditure on mx missiles=yes; Federal subsidy to nuclear power stations=yes; Subsidy to national parks in Alaska=yes; Fair housing bill=yes; Limit on Pac contributions=yes; Limit on food stamp program=no, Federal help to education=no; StateFrom=north east; State Population=large; Occupation=unknown; Cut in social security spending=no, Federal help to Chrysler corp=not-registered.

By expressing elementary statements in the example as conditions, and linking conditions by conjunction, the examples can be re-expressed as decision rules. Thus, decision rules and examples formally differ only in the degree of generality.

3. Generating decision trees from decision rules

3.1. Related research

Decision trees are normally generated from examples. The essential aspect of any such method is the *attribute selection criterion* that is used for choosing attributes to be assigned to the nodes of the tree being built. Criteria for that purpose include the entropy reduction measure (e.g., Quinlan, 1979), the gini index of diversity (Breiman et al., 1984), and many others (Cestnik and Karalic, 1991; Mingers, 1989a).

An early algorithm for generating decision trees from examples was proposed by Hunt, Marin and Stone (1966). This algorithm was subsequently modified by Quinlan (1979, 1983), and improved and/or applied by many authors to a variety of learning problems (e.g., Quinlan, 1983, 1985; Breiman et al., 1984).

Later, Quinlan (1986) and Bratko and Kononeko (1987) extended the method to handle also data with noise (by pruning). Such a method consists of two phases: the creation of an initial decision tree, and "tree pruning," done by removing subtrees with small statistical validity, and replacing them by leaf nodes. More recently, pruning has also been used for simplifying decision trees even for noiseless problems (Cestnik and Bratko, 1991). Pruning decision trees improves their simplicity, but reduces their predictive accuracy on the training examples. Quinlan (1990) proposed also a method to handle the "unknown attribute value" problem, by exploring probabilities of an example belonging to different classes.

The AQDT-1 method proposed here generates decision trees from decision rules. As mentioned earlier, decision rules used in this method are obtained by an AQ-type inductive learning program (AQ15 or AQ17-DCI). One problem in developing a method for generating decision trees from decision rules is to design an attribute selection criterion that is based on the properties of *the rules*, rather than of the *training examples*. A decision rule normally describes a number of possible examples. Only some of them are examples that have actually been observed, i.e., training examples. An attribute selection criterion needs to analyze the role of each attribute in the rules. It cannot be based on counting the numbers of training examples "covered" by each attribute value, as done in learning decision trees from examples, because training examples are assumed to be unavailable.

Another problem in learning decision trees from decision rules stems from the fact that decision rules constitute a more powerful knowledge representation than decision trees. They can directly represent a description in an arbitrary disjunctive normal form, while decision trees can represent directly only descriptions in the "disjoint" disjunctive normal form. In such descriptions, all conjunctions are mutually logically disjoint. Therefore, when transforming a set of arbitrary decision rules into a decision tree, one faces an additional problem of handling logically intersecting rules.

The solution of the first problem (attribute selection) in the AQDT-1 system is based on the earlier work by Michalski (1978), which introduced a general method for generating decision trees from decision rules. The method aimed at producing decision trees with the minimum number of nodes or the minimum cost (where the "cost" was defined as the total cost of classifying unknown examples, given the cost of measuring individual attributes and the expected probability distribution of examples of different decision classes). The method proposed several attribute selection algorithms of increasing power (the nth order cost estimates, $n = 1, 2, \ldots$), and analyzed two specific criteria, MAL and DMAL, for selecting the "optimal" attribute for a node in the tree, based on rule properties. The MAL criterion ("minimizing added leaves") seeks an attribute that minimizes the estimated number of additional nodes in the decision tree being generated over a hypothetical minimal decision tree. The AQDT-1 uses an approximate version of this criterion (the "attribute dominance"). The DMAL criterion ("dynamically minimizing added leaves") is based on the similar principle as MAL, but is more powerful, because it takes into consideration additional information. DMAL is more difficult to implement, and the current version of the program does not include it.

3.2. The AQDT-1 method for attribute selection

In its basic form (the default version), the attribute selection method employed in AQDT-1 seeks the simplest (i.e., with the minimum number of nodes) decision tree. In a more general form, the method seeks the minimum-cost decision tree, that is, a tree that minimizes the overall cost of making classification decisions. If measuring different attributes involves different costs, these costs are taken into consideration. If some decision classes occur much more frequently than others, then the method should favor measuring first the attributes occurring in the rules for the most frequent classes. The issue of producing the simplest trees is addressed first, and then it is shown how the method is modified to address the minimum cost requirement.

The method for choosing attributes on the basis of rule properties is based on an *attribute utility* ranking that is based on three elementary criteria: (1) *disjointness*—that captures the attribute effectiveness in discriminating among decision rules of different decision classes, (2) *dominance*—that measures the attribute relevance by counting the number of rules that contain the attribute, and (3) *extent*—that measures the number of different attribute values present in the rules.

The disjointness of an attribute is defined as the sum of the *attribute class disjointness*—the disjointness of the attribute for each decision class. Suppose decision classes are $C_1, C_2, \ldots, C_m$, and decision rulesets for these classes have been determined. Given attribute A, let $V_1, V_2 \ldots, V_m$, denote sets of the values of attribute A that are present in rulesets for classes $C_1, C_2, \ldots, C_m$, respectively.

If a ruleset for some class, say, C_a contains a rule that does not involve the attribute A, than V_a is the set of all possible values of A (the domain of A).

DEFINITION 3.1. The *degree of class disjointness, $D(A, C_i)$ of attribute A for the ruleset of class C_i*, is the sum of the degrees of disjointness, $D(A, C_i, C_j)$, between the ruleset for C_i and rulesets for C_j, $j = 1, 2, \ldots, m, j \neq i$. The degree of disjointness between the ruleset for C_i and the ruleset for C_j is defined by

$$D(A, C_i, C_j) = \begin{cases} 0, & \text{if } V_i \subseteq V_j \\ 1, & \text{if } V_i \supset V_j \\ 2, & \text{if } V_i \cap V_j \neq \phi \text{ or } V_i \text{ or } V_j \\ 3, & \text{if } V_i \cap V_j = \phi \end{cases}$$

where ϕ denotes the empty set.

DEFINITION 3.2. The disjointness of attribute A for evaluating a given set of decision rules is the sum of the degrees of class disjointness for each decision class:

$$D(A) = \sum_{i=1}^{m} D(A, C_i) \text{ where } D(A, C_i) = \sum_{i=1, j\neq i}^{m} D(A, C_i, C_j) \tag{1}$$

The disjointness of an attribute ranges from 0, when the attribute values in rulesets of different classes are all the same, to $3{*}m{*}(m-1)$, when every ruleset of a given class contains a different set of the attribute values. Selecting an attribute with the maximum possible disjointness produces a node of the decision tree whose children can be immediately assigned decision classes.

The second elementary criterion, dominance, prefers attributes that appear in large number of rules, as this indicates their high relevance for discriminating among ruleset of given decision classes. Since some conditions in the rules have values linked by internal disjunction, counting such rules directly would not reflect properly their relevance. Therefore, for computing the dominance, the rules are counted as if they were converted to elementary rules that do not have internal disjunction. Such a conversion is done by "multiplying out" the condition parts of the rules containing internal disjunction. For example, the condition part $[x3 = 1 \vee 3]\&[x4=1]$ is "multiplied out" to two rules with condition parts $[x3=1]\&[x4=1]$ and $[x3=3]\&[x4=1]$.

The third elementary criterion, extent, prefers attributes with fewer values in the rules. Nodes that are assigned such attributes will have a smaller fan out. When attributes are continuous, they are quantized into discrete units representing ranges of values.

The above three elementary criteria are combined into one general attribute ranking measure using the "lexicographic evaluation functional with tolerances"

(LEF) (Michalski, 1973). The LEF allows to combine the elementary criteria in different ways. In the default combination, the LEF is defined as

⟨Disjointness, $\tau 1$; Dominance, $\tau 2$; Extent, $\tau 3$⟩

where $\tau 1$, $\tau 2$, $\tau 3$ are tolerance thresholds.

The above LEF ranks attributes this way. First, attributes are evaluated on the basis of their disjointness. If two or more attributes share the same top score, or their scores differ less than the assumed tolerance threshold $\tau 1$, the method evaluates these attributes using the second (dominance) criterion. If again two or more attributes share the same top score or their scores differ less than the tolerance threshold $\tau 2$, then the third criterion, extent, is used. If there is still a tie, the method selects the "best" attribute randomly. The system tries for maximize disjointness and dominance of an attribute, while minimizing the extent.

In the nondefault version of the attribute selection criterion, the above definition changes as follows. If the costs of measuring attributes are known and are nonuniform, then the attribute selection method seeks the *minimum cost* attribute. If an attribute cannot be measured, then its cost is assumed to be infinite.

The attribute ranking criterion is defined by a LEF consisting of four elementary criteria:

⟨Cost, $\tau 1$; Disjointness, $\tau 2$; Dominance, $\tau 3$; Extent, $\tau 4$⟩

where the Cost denotes the evaluation cost of an attribute and is to be minimized, while other elementary criteria are treated the same way as in the default version. Thus, attributes with minimum or comparable costs (within the defined tolerance) will have the preference over other attributes. Finally, if one wants to take into consideration a nonuniform distribution of examples of different classes, then the attribute ranking criterion is modified by differently evaluating the degree of attribute disjointness. Namely, the previously defined degree of disjointness for a given class is multiplied by the frequency of examples of this class.

3.3. *The AQDT-1 algorithm*

AQDT-1 constructs a decision tree from decision rules by recursively selecting at each step the "best" attribute according to the above-described attribute ranking measure, and assigning it to the new node. The process stops when the algorithm creates terminal branches that are assigned decision classes.

To facilitate such a process, the system creates a special data structure for each concept description (ruleset). This structure has fields such as the number of rules, the number of conditions in each rule, and the number of attributes in the rules. The system also creates an array of attribute descriptions. Each attribute description contains the attribute's name, domain, type, the number of legal values, a list of the values, the number of rules that contain that attribute, and values of that attribute for each rule. The attributes are arranged in the array in the a lexicographic order, first, in the descending order of the number of rules that contain that attribute, and second, in the ascending order of the number of the attribute's legal values.

The system can work in two modes. In the *standard* mode, the system generates standard decision trees, in which each branch has a specific attribute value assigned. In the *compact* mode, the system builds a decision tree that may contain:

(A) "or" branches, i.e., branches assigned an internal disjunction of attribute values, whenever it leads to simpler trees. For example, if a node assigned attribute A has a branch marked by values "1 v 2," then the control passes along this branch whenever A takes value 1 or 2. The program creates "or" branches on the basis of the analysis of the value sets V_i, while computing the degree of attribute disjointness.

(B) nodes that are assigned *derived* attributes, that is, attributes that are certain logical or mathematical combinations of the original attributes. To produce decision trees with derived attributes, the input decision rules are generated by program AQ17-DCI (rather than AQ15). The AQ17-DCI rules may contain conditions involving attributes constructed by the program, rather than those originally given. If a AQ17-DCI discovers a particularly useful attribute, then the decision rules and consequently the derived from them decision trees can be significantly simplified (compare, for example, decision tree in Figure 9b with the one in Figure 11).

To generate decision trees from rules, the method uses characteristic descriptions generated in the "DC" (disjoint cover) mode of the AQ15 (or AQ17-DCI) program. The reason for using characteristic descriptions is that they offer a greater choice of attributes in the process of building a decision tree, and this may lead to simpler decision trees. The reason for disjoint rulesets is that they are more suitable for building decision trees, as the latter are equivalent to sets of logically disjoint descriptions.

Assume that the input contains characteristic descriptions of the given decision classes. The description of each class is in the form of a ruleset. Assume that this set is the initial *ruleset context*.

Step 1. Evaluate each attribute occurring in the ruleset context using the LEF attribute ranking measure. Select the highest ranked attribute. Suppose it is attribute A.

Step 2. Create a node of the tree (initially, the root, afterwards, a node attached to a branch), and assign to it the attribute A. In the standard mode, create as many branches from the node, as there are legal values of the attribute A, and assign these values to the branches. In the compact mode, create as many branches as there are disjoint value sets of this attribute in the decision rules, and assign these sets to the branches.

Step 3. For each branch, associate with it a group of rules from the ruleset context that contain a condition satisfied by the value(s) assigned to this branch. For example, if a branch is assigned values i of attribute A, then associate with it all rules containing condition $[\mathbf{A} = \mathbf{i} \vee \ldots]$. If a branch is assigned values $i \vee j$, then associate with it all rules containing condition $[\mathbf{A} = \mathbf{i} \vee \mathbf{j} \vee \ldots]$. Remove from the rules these conditions. If there are rules in the ruleset context that do not contain attribute A, add these rules to all rule groups associated with the branches stemming from the node assigned attribute A. (This step is justified by the consensus law: $[x{=}1] \equiv \{ [x{=}1]\ \&\ [y{=}a] \vee [x{=}1]\ \&\ [y{=}b] \}$, assuming that a and b are the only legal values of y.) All rules associated with the given branch constitute a ruleset context for this branch.

Step 4. If all the rules in a ruleset context for some branch belong to the same class, create a leaf node and assign to it that class. If all branches of the trees have leaf nodes, stop. Otherwise, repeat steps 1 to 4 for each branch that has no leaf.

The algorithm was implemented in the C language on the SPARC II machine. The average running time of the algorithm, for the experiments presented in this paper, was below one second.

3.4. *An example illustrating the algorithm*

The following simple example illustrates the AQDT-1 algorithm. Suppose there are three decision classes, $C1$, $C2$, & $C3$, described by the AQ15-derived ruleset shown in Figure 2.

```
{C1  <=[x1=2] & [x2=2]             , C1<=[x1=3] & [x3 = 1 v 3] & [x4=1] }
{C2  <=[x1 = 1 v 2] & [x2 = 3 v 4] , C2<=[x1=3] & [x3 = 1 v 2] & [x4=2] }
{C3  <=[x1=1] & [x2=1]             , C3<=[x1=4] & [x3 = 2 v 3] & [x4=3] }
```

Figure 2. Rules used for illustrating the algorithm.

Suppose that these rules constitute the initial ruleset context. Table 1 presents information on these rules and the values of elementary criteria computed for all attributes. For each class, the row marked "Values" lists values occurring in the ruleset for this class. For evaluating the disjointness of an attribute, say **A**, each rule in the ruleset above that does not contain attribute A is characterized as having an additional condition $[\mathbf{A} = a \ \vee b \ldots]$, where $a, b, \ldots$ are all legal values of **A**.

The row "Class disjointness" specifies the class disjointness for each attribute. The attribute $x1$ has the highest disjointness (11), and is assigned to the root of the tree. For simplicity, assume the tolerances for each elementary criterion equal 0.

From the rules in Figure 2, we can also determine disjoint groupings of

Table 1. Determining values of the selection criteria for each attribute.

		Attributes			
Class		$x1$	$x2$	$x3$	$x4$
C1	Values	2, 3	1..4	1..3	1..3
	Class disjointness	3	0	0	0
C2	Values	1, 2, 3	1..4	1..3	1..3
	Class disjointness	3	0	0	0
C3	Values	1, 4	1..4	1..3	1..3
	Class disjointness	5	0	0	0
Attribute Disjointness		11	0	0	0
Attribute Dominance		12	6	6	6
Attribute Extent		4	4	3	3

attribute values used for the compact mode of the algorithm. This done as follows: (1) determine for each attribute the sets of values that the attribute takes in individual decision rules, and remove those value sets that subsume other value sets. The remaining value sets are assigned to branches stemming from the node marked by the given attribute. For example, $x1$ has the following value sets in the individual decision rules: {2}, {3}, {1, 2}, {1}, and {4} (Figure 2). Value set {1, 2} is removed as it subsumes {2} and {1}. In this case, branches are assigned individual values of the domain of $x1$. For attribute $x2$, the value sets are {1}, {2}, {3, 4}, and {1, 2, 3, 4}. In this case, branches are assigned value sets: {1}, {2} and {3, 4}.

Attribute $x1$ ranks highest (as it is the highest disjointness), and is assigned to the root of the tree. Four branches are created each one is corresponding to one of $x1$ possible values. Since all rules containing {$x1$=4} belong to class *C3*, the branch marked by 4 is ended by a leaf *C3*. Rules containing other values of $x1$ belong to more than one class.. This process is repeated for each subset of rules until the decision tree is completed. Figure 3a shows a conventional decision tree, and Figure 3b a compact decision tree learned from these rules. For combinations of attribute values that do not lead to any leaf, the decision tree assigns no decision ("unknown" decision). Figure 4a shows the diagrammatic visualization of the decision rules and Figure 4b of the derived decision tree.

Each diagram in Figure 4 consists of cells representing one combination of attribute values. Attributes and their legal values are shown on scales surrounding the diagram (e.g., the horizontal scale for $x1$ shows values 1, 2, 3 and 4). Rules correspond to collections of cells in the intersection of the rows and columns

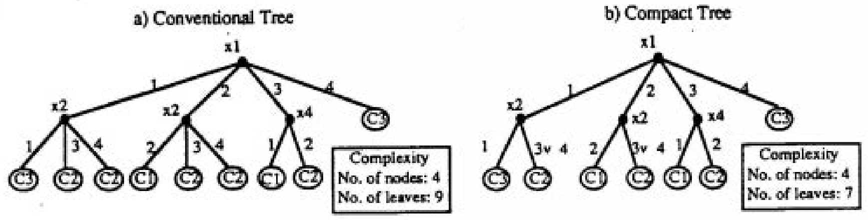


Figure 3. Decision tree generated for the rules in Figure 2.

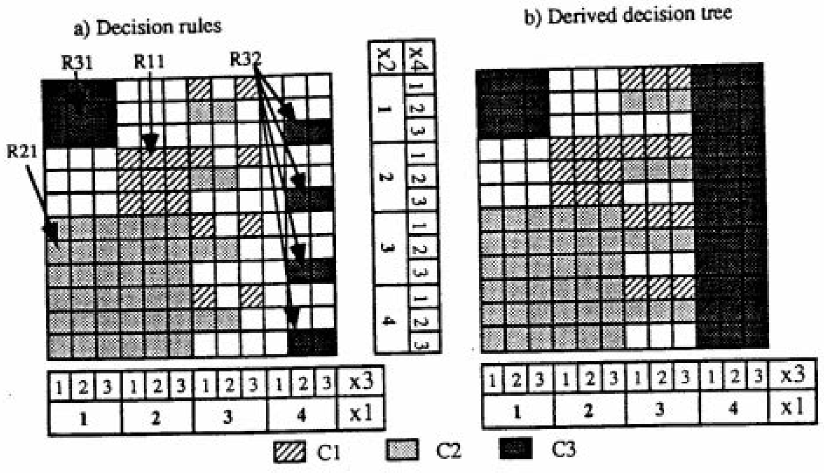


Figure 4. A diagrammatic visualization of the decision rules and the derived decision tree.

corresponding to the conditions in the rules. The shaded areas correspond to individual rules. Rules of the same class have the same type of shading. Empty cells correspond to combination of attribute values not assigned to any class. For illustration, collections of cells corresponding to some of the initial rules are marked $R11$, $R21$, $R31$, and $R32$. $R11$ denotes the first rule of Class $C1$, i.e., $[x1=2]$ & $[x2=2]$; $R21$ denotes the first rule of class $C2$, i.e., {$[x1 = 1 \vee 2]$ & $[x2 = 3 \vee 4]$, $R31$ the first rule of class $C3$, i.e., $[x1=1]$ & $[x2=1]$; and denotes $R32$ denotes the second rule of class $C3$, i.e., $[x1=4]$ & $[x3 = 2 \vee 3]$ & $[x4=3]$ }. Comparing diagrams in Figure 4a and 4b, one can see that the derived decision tree represents a slightly more general description of Concepts $C1$, $C2$, and $C3$ than original rules.

Let us assume that determining the value of $x2$ is impossible, which is indicated to AQDT-1 by assigning very high cost to $x2$. The algorithm assigns again attribute

$x1$ to the root of the tree. When $x1$ takes value 1 or 2, it is impossible to assign a specific decision without measuring attribute $x2$. For the value 1 of $x1$, the decision class can be $C2$ and $C3$ (see the diagram in Figure 4a), and for the value 2 of $x1$, the decision class can be $C1$ or $C3$. However, for the value 3 of $x1$, one can make a specific decision after measuring attributes $x4$. For the value 1 of $x4$ decision is $C1$, and for the value 2 of $x4$ the decision is $C2$. Figure 5 shows the obtained decision tree. Such a tree is called indeterminate, because some of its leafs are assigned a disjunction of two or more class names. These leafs indicate situations in which without measuring $x2$ one cannot make a specific decision.

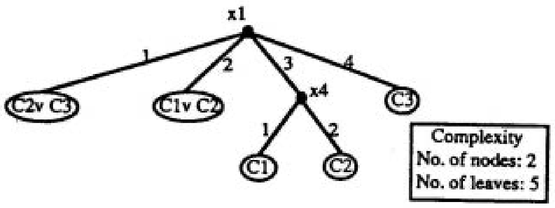


Figure 5. The indeterminate decision tree generated from rules in Figure 2 under the assumption of unavailability of $x2$.

Let us now suppose that $x1$, which was determined as the highest ranked attribute, cannot be measured. Each of the remaining attributes, $x2$, $x3$ and $x4$, has disjointness 0 and dominance 6. The extent of $x3$ and $x4$ is 3 (the number of difference values in rules), and of $x2$ is 4. The algorithm selects randomly an attribute between $x3$ and $x4$, and assigns it to the root of the tree. Suppose it is $x4$. After continuing the algorithm, the tree in Figure 6 is obtained. The nodes that are assigned one class indicate situations in which it is possible to make a specific decision·without knowing the value of attribute $x1$. Figure 7 shows a diagrammatic visualization of each class individually without using $x1$. The shades areas in these diagrams that do not have common cells indicate situations (combinations of attribute values) for which it is possible to make a specific decision without knowing the value of $x1$. If $x1$ can be measured, but measuring is very expensive, then the decision tree will have a node assigned $x1$ as far as possible from the root.

3.5. *Experiment 1*

This experiment illustrates the application of AQDT-1 to the so-called MONK-1 problem (Thrun, Mitchell & Cheng, 1991). This problem is to learn a concept from 124 training examples (62 positive and 62 negative) expressed in terms

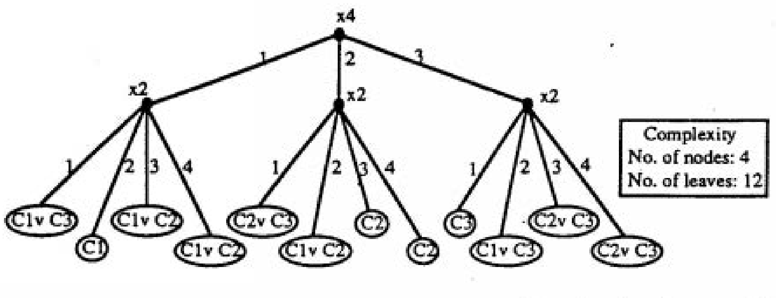


Figure 6. An indeterminate decision tree generated from rules in Figure 2 under the assumption that $x1$ cannot be measured.

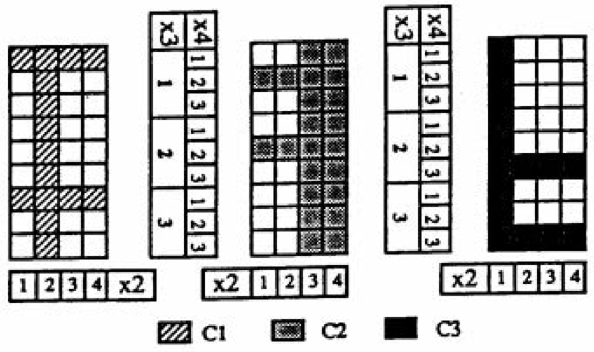


Figure 7. A diagrammatic visualization of the individual classes and the decision trees that do not contain $x1$ node.

of 6 multivalued attributes. These training examples constitute 30% of the all possible examples (432), thus the "density" of the training examples is relatively high. For comparison, the well-known program C4.5 for learning decision trees from examples was also applied to this same problem.

It was assumed that both programs should produce a complete and consistent decision tree with regard to the training examples, i.e., a decision tree that gives 100% correct performance on the training examples).

The C4.5 program has the capability of generating a decision tree for a "window" of examples (a randomly selected subset of the training examples). It starts with a randomly selected window, generates a trial tree, adds some unclassified objects, and continues until its all training examples are classified correctly, or it cannot produce a better tree. This entire process is repeated 10 times. The results presented here and for all experiments are the best result obtained when running C4.5 with both default windows (maximum of 20% and twice the square root of number of examples) and 100% windows (i.e., the entire set of training examples).

The AQDT-1 program, running in the conventional mode and with the optimality criterion set to minimize the number of nodes, produced a decision tree with 41 nodes. The C4.5 program did not produce a consistent and complete decision tree when run with its default window size (max. of 20% and twice the square root of number of examples) nor with 100% window size. After 10 trials with different window sizes, we succeeded in making C4.5 produce the same decision tree as AQDT-1 (using the window size of 72.5%). The tree is presented in Figure 8.

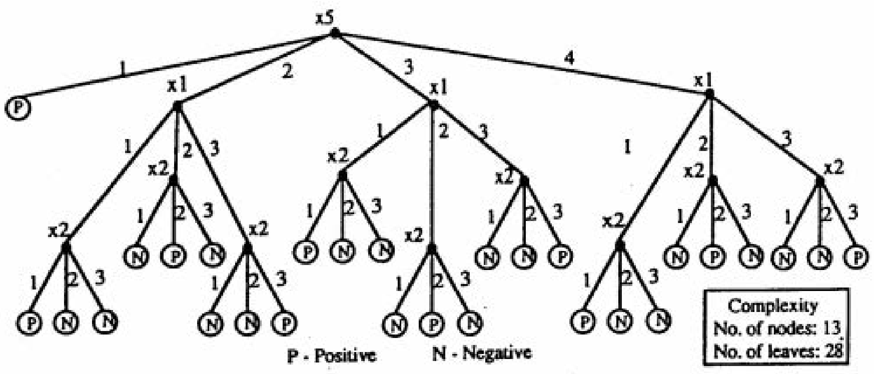


Figure 8. The decision tree for the MONK-1 problem generated both by AQDT-1 (in conventional mode) and C4.5.

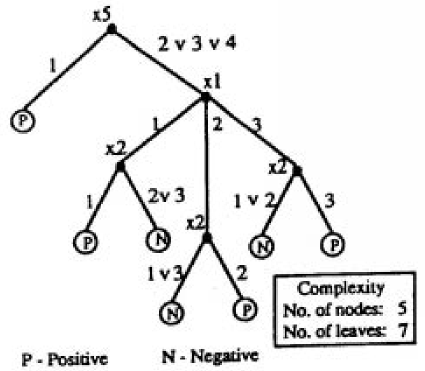
By running AQDT-1 in "compact" mode, a simpler decision tree was produced (Figure 9a). In the final experiment, we used AQ17-DCI (Bloedorn and Michalski, 1991a, b) to derive decision rules. These rules were

$$\text{Pos} <= [x5 = 1] \vee [x1 = x2] \quad \text{and} \quad \text{Neg} <= [x5 \neq 1] \ \& \ [x1 \neq x2]$$

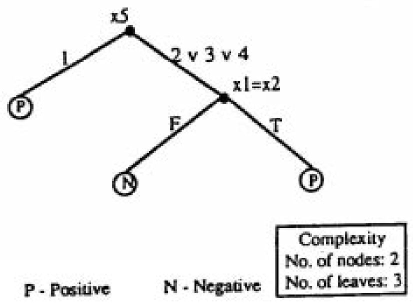
From these rules, the system produced the compact decision tree presented in Figure 9b. It should be noted that decision trees in Figures 8, 9a, and 9b are all logically equivalent, and they all have 100% prediction accuracy on testing examples (which means that they represent exactly the target concept).

3.6. *Experiment 2*

To test AQDT-1 on a real-world problem, it was applied to learning patterns in the U.S. Congress voting records of 1981. Again, for comparison, C4.5 was also



(a) Compact decision tree derived from AQ15 rules.



(b) Compact decision tree derived from AQ17-DCI rules.

Figure 9. Compact decision trees generated by AQDT-1 for the MONK-1 problem.

applied to the same problem. There are two decision classes: the "Democratic Voting Pattern" and the "Republican Voting Pattern." Each voting record of a Democrat or a Republican is described in terms of 19 multivalued attributes. Our experiment used 51 voting records (31 Democratic and 20 Republican).

The AQ15 inductive learning program generated four rules for the "Democratic Voting Pattern," and seven rules for the "Republican Voting Pattern." Only 10 of 19 original attributes were used in the learned rules. The AQDT-1 program, running in the conventional mode and with the optimality criterion set to minimize the number of nodes, produced a decision tree with 20 nodes. The prediction accuracy of the tree on the testing examples was 91.8%.

For comparison, C4.5, was also run on exactly the same data. C4.5 produced a decision tree with 23 nodes, and its prediction accuracy on the same testing examples was 85.7%. (Both programs run under the assumption that they will produce a complete and consistent decision tree with regard to the training examples, i.e., a decision tree that gives 100% correct recognition on the training examples).

To provide more details on this experiment, Table 2 presents attributes involved in the decision rules, and their legal values (domains).

For simplicity, original symbolic values have been mapped into isomorphic numerical values. These numerical values correspond to the symbolic values listed in Table 2.

Figure 10 presents decision trees generated by AQDT-1 for the above Congressional Voting problem (1981). Figure 10a shows a conventional decision tree, generated from AQ15 rules, and Figure 10b shows a compact decision tree, generated from AQ17-DCI rules. The compact decision tree contains some nodes

Table 2. The U.S. Congressional voting attributes and their legal value sets.

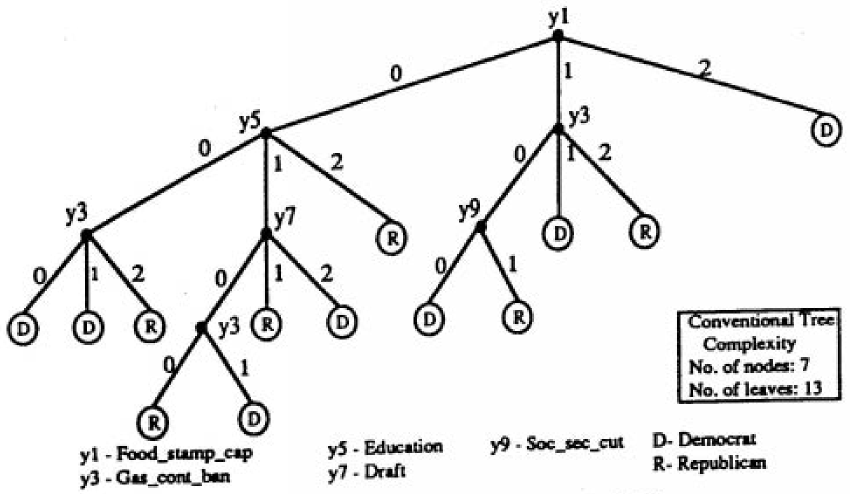
Attribute		**Legal values**	
y1–Food stamp cap	0–no	1–yes	2–not registered
y2–Occupation	0–known	1–unknown	
y3–Gas cont ban	0–no	1–yes	2–not registered
y4–Income	0–low	1–medium	2–high
y5–Education	0–no	1–yes	2–not registered
y6–Chrysler	0–no	1–yes	2–not registered
y7–Draft	0–no	1–yes	2–not registered
y8–State	0–northwest	1–northeast	2–not registered
y9–Soc sec cut	0–no	1–yes	2–not registered
y10–Alaska parks	0–no	1–yes	2–not registered
y11–Wind tax limit	0–no	1–yes	2–not registered
y12–Nicaragua ban	0–no	1–yes	2–not registered
y13–Fair housing	0–no	1–yes	2–not registered
y14–Nuc power	0–no	1–yes	2–not registered
y15–Pac limit	0–no	1–yes	2–not registered
y16–Mx cut	0–no	1–yes	2–not registered
y17–Osha cut	0–no	1–yes	2–not registered
y18–Hosp cost cont	0–no	1–yes	2–not registered
y19–Population	0–small	1–medium	2–large

that are assigned constructed attributes: $y20$, $y21$ and $y22$. These attributes represent simple mathematical relations on the initial attributes. The attribute $y20$ is defined by the relation $y7+y3 = 1 \vee 2$. (The attribute takes value T (true) whenever the sum of the numeric values of $y7$ and $y3$ equals one or two, and value F (false), otherwise.) Attribute $y21$ is defined by the relation $y12+y9 \leq 3$, and attribute $y22$ is defined by the relation $y12-y4 = 0 \vee 1$.

For comparison, Figure 11 presents a decision tree generated by C4.5 for the same problem.

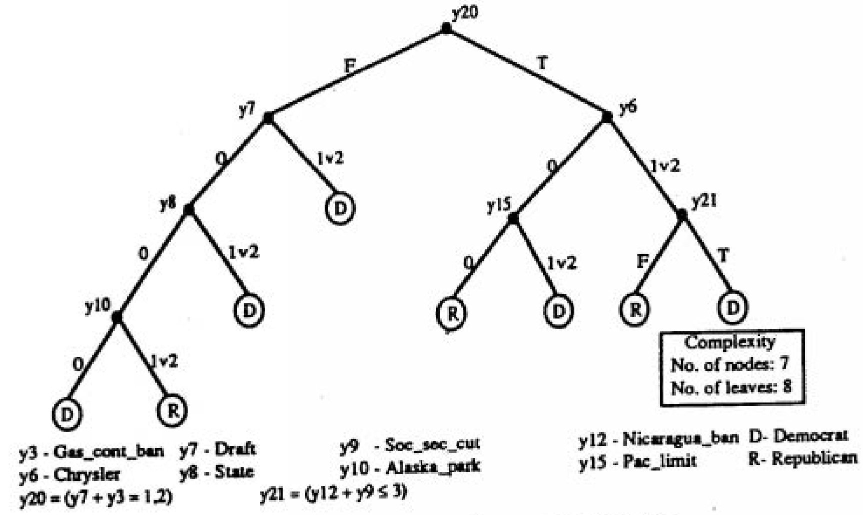
Comparing decision trees in Figures 10 and 11, one can notice that the trees generated from decision rules (conventional and compact) had a higher predictive accuracy (91.8% vs. 85.7%) on testing examples and were simpler that the tree generated directly from examples by C4.5.

3.7. *Pruning AQDT-1 decision trees*



The predictive accuracy on testing examples is 91.8%.

(a) A conventional decision tree.



The predictive accuracy on testing examples is 91.8%.

(b) A compact decision tree.

Figure 10. Decision trees obtained by AQDT-1 for the Congressional Voting (1981) problem.

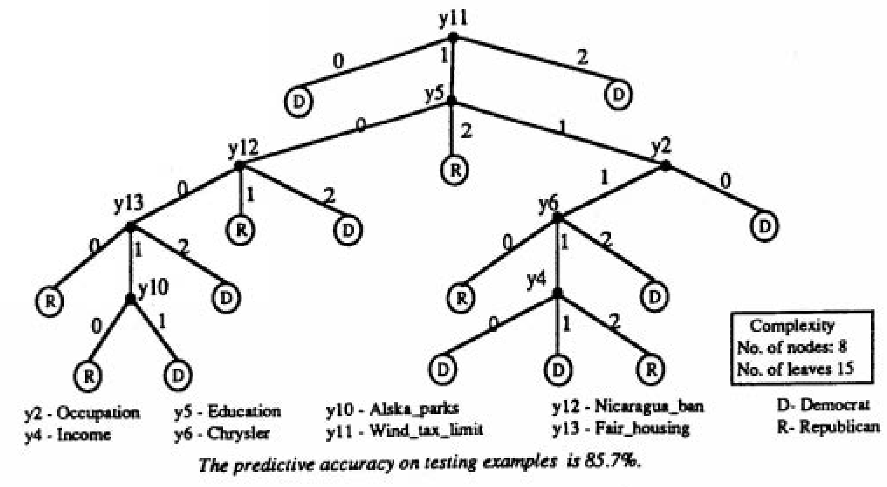
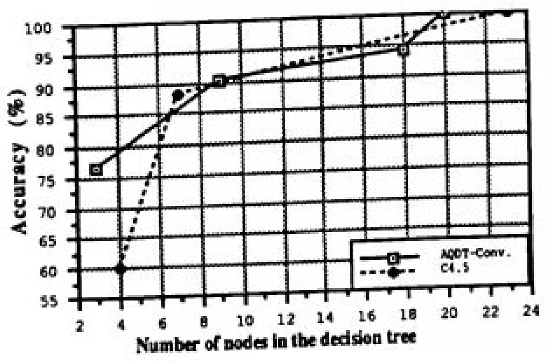


Figure 11. Decision tree obtained by C4.5 for the Congressional Voting (1981) problem.

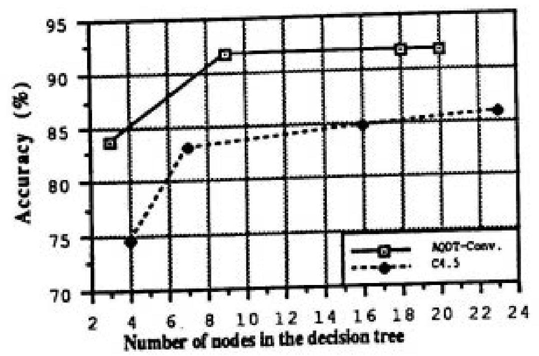
When input data may contain errors (noise), it is often useful to "prune" parts of the (consistent and complete) decision tree that have small statistical significance. Such pruning protects the tree from over fitting. Various approaches to such "tree pruning" have been described in (Mingers, 1989b; Breiman et al., 1984; Quinlan, 1986, 1987; Niblett and Bratko, 1986; Cestnik and Karalic, 1991; Clark and Niblett, 1987; Smyth et al., 1990; Cestnik and Bratko, 1991). These approaches differ in the criteria for deciding whether or not to prune the tree at some level. A comparison of these pruning approaches is in (Mingers, 1989b).

When decision trees are generated from decision rules, it is better to prune the rules before they are used for determining the tree. Pruning decision rules is done on basis of the "rule strength." The rule strength is characterized by its *t-weight* and *u-weight*. The *t-weight (total-weight)* of a rule for some class is the number of examples of that class covered by the rule. The *u*-weight *(unique-weight)* of a rule for some class is the number of examples of that class covered only by this rule.

The proposed method follows ideas presented in (Michalski et al., 1986), namely, to prune rules of small strength, specifically, the rules with the *t*-weight or the *u*-weight below certain threshold. So pruned rules are used for determining a decision tree. For example, in the Congressional voting domain, four rules characterize Democratic records and seven rules characterize Republican records. Of these 11 rules in toto, five rules have *t*-weight equal 1. These five rules (one for the Democratic vote class and four for the Republican vote class) are pruned (truncated). The remaining rules are used for creating decision tree using the



(a) For training examples



(b) For testing examples

Explanation: The complete (unpruned) tree generated by AQDT-1 had 20 nodes, and the one generated by C4.5 had 23 nodes.

Figure 12. The dependence of the predictive accuracy of the decision tree on the size of the tree after pruning for the Congressional Voting problem (1981).

AQDT-1 algorithm. The decision tree learned this way had 18 nodes (vs. 20 nodes in the decision tree obtained from the unpruned rules), and the prediction accuracy of 91.8% on the testing examples (no change), and 94.1% on training examples (vs. 100%). The results of further pruning are presented in Figure 12.

For comparison, C4.5 generated a consistent decision tree with 23 nodes before pruning (Figure 11). After pruning it had 7 nodes and prediction accuracy of 88.2% on the training examples and 83.1% with testing examples. Figure 12 presents the dependence of the predictive accuracy on the size (the number of nodes) of the pruned tree for the Congressional voting problem (1981). The experiment was done using training examples (Figure 12a) and testing examples (Figure 12b).

4. Comparing learning behavior of AQDT-1 and C4.5

This section presents experiments comparing the learning behavior of AQDT-1 and C4.5 when learning from the training data sets of varying size. These experiments were performed for the U.S. Congressional Voting (1984) problem, and the MONK-1 problem (described earlier).

There are many ways to test the performance of learning systems, such as hold-out, leave-one-out, and cross-validation (e.g., Arciszewski, Dybala and Wnek, 1992). In comparing learning systems, one should apply them to training datasets of different sizes. In this experiment, we applied a variation of the hold-out method, in which the training set of examples varies from experiment to experiment.

Experiments were performed for two problems. One was the U.S. Congressional Voting problem (1984). Each example was described in terms of 16 attributes. There were two decision classes, and total 216 examples. The experiments tested the change in the number of nodes and the predictive accuracy with varying the number of training examples used for generating a decision tree by AQDT-1 and C4.5. The experiment was done with C4.5 using two window options, the default option (maximum of 20% the number of examples and twice the square root the number of examples), and with 100% window size (one trial per each setting). In the Congressional Voting (1984) problem, the sizes of the set of training examples were 8%, 16%, 24%, 31%, 39% and 52% of the total number of training examples (216 examples in total; half of the examples were in one class and the second half in the other class).

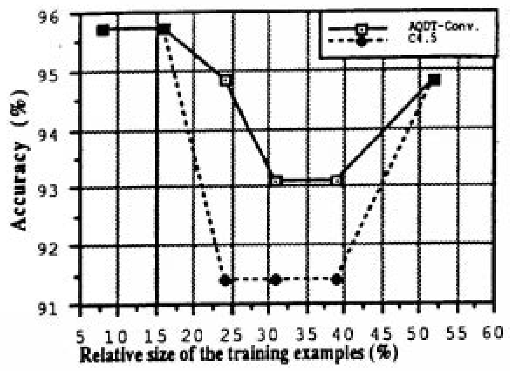
Another problem used for testing was the MONKs' first problem. The MONK-1 data had in toto 432 examples, half of them representing one decision class, and the other half representing the second class. In Monk-1 problem, the sizes of the set of training examples were 5%, 10%, 15%, 20%, 25% and 34% of the total set of training examples (100% stands for 432 examples).

Figures 13a and b show the results graphically for the Congressional voting record (1984). Figures 14a and b show the results graphically for the MONK-1 problem.

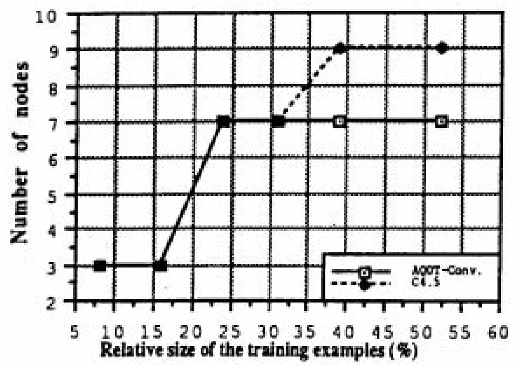
Table 3 and Figures 13 and 14 indicate that AQDT-1 generated decision trees had a higher accuracy and were simpler than decision trees produced by C4.5. Also, the variations of the size of the AQDT-1's trees with the change of the size of training example set were somewhat smaller.

5. Conclusion

The paper argues for generating decision trees from decision rules rather than from examples, as has been done conventionally. The reason for the proposed approach is that it is easier to generate a decision tree tailored to any given



(a) The accuracy of the decision tree as a function of the size of the set of training examples.

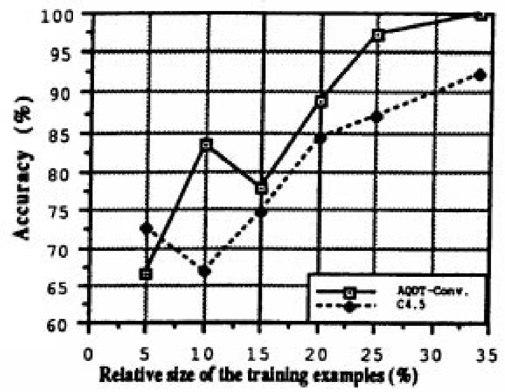


(b) The size of the decision tree as a function of the size of the set of the training examples.

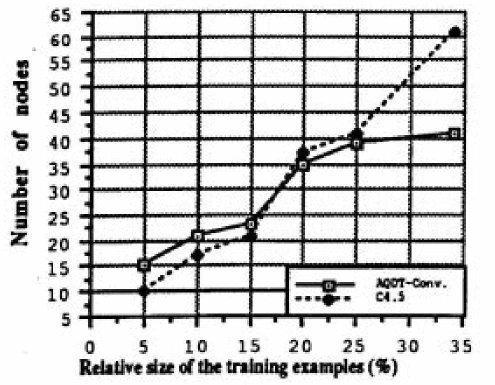
Figure 13. The predictive accuracy of the AQDT-1 and C4.5 decision trees for the Congressional Voting 1984 problem as a function of the size of the set of training examples.

decision making situation from rules than to modify a decision tree once created. The presented method, AQDT-1, efficiently determines a decision tree from decision rules generated by the AQ15 or AQ17-DCI inductive learning programs. The paper introduced also the idea of a "compact" decision tree, in which branches can be associated with a set of attribute values and nodes with attributes derived from initial attributes via constructive induction. Compact decision trees generated by the method were consistently simpler than conventional trees.

The main difference between determining trees from decision rules and determining them from examples is in the attribute selection criterion. In the former case, the attribute selection function needs to evaluate the role of attributes in



(a) The accuracy of the decision tree as a function of the size of the set of training examples.



(b) The size of the decision tree as a function of the size of the set of the training examples.

Figure 14. Comparing decision trees for MONK-1 data generated by C4.5 and AQDT-1.

the rules, while in the latter case, the criterion evaluates the way the attribute splits the training examples. The AQDT-1 method uses an attribute ranking criterion composed of three elementary criteria: the dominance and the extent of an attribute in the decision rules. Further research will investigate other attribute selection criteria for generating decision trees from rules.

A disadvantage of the proposed method is that it requires a generation of decision rules first. However, there are efficient rule learning systems. The AQDT-1 method uses the AQ15 or AQ17-DCI learning programs for this purpose. Since the method is independent on what rules are used as input, it could potentially be applied also with other decision rule learning systems, or with

Table 3. A tabular summary of the predictive accuracies of decision trees obtained by AQDT-1 and C4.5 (as shown in Figures 13 and 14).

Problem	**Relative size of training data (%)**	**Experiment**			
		Accuracy (%)		**Number of nodes**	
		C4.5	AQDT	C4.5	AQDT
Congressional	8	95.7	95.7	3	3
Voting 1984	31	91.4	93.1	7	7
	52	94.8	94.8	9	7
	5	72.6	66.6	10	15
MONK-1	20	84.2	88.8	37	35
	34	92.4	100.0	61	41

decision rules acquired from an expert.

One advantage of the proposed method is that it allows one to efficiently determine a decision tree that is optimized for a given decision making situation. For example, when some attribute is difficult to measure, the method creates a decision tree that shows in which situations measuring this attribute can be avoided. The method is quite efficient, and the time of determining a decision tree from decision rules in the cases we investigated was negligible. Therefore, it is easily to experiment with different criteria for tree generation in order to obtain the most desirable tree.

Another advantage is that decision trees obtained this way tend to be simpler and have higher predictive accuracy. In the experiments done so far, the AQDT-1 generated decision trees have consistently outperformed those generated by C4.5 program both in terms of predictive accuracy and simplicity of the decision tree. In the experiments, the program was applied to a made-up problem and a real-world problem.

Acknowledgments

The authors thank Eric Bloedorn, Mike Hieb, Ken Kaufman, Mark Maloof, Haleh Vafaie, and Janusz Wnek for valuable comments and suggestions. This research was conducted in the Center for Artificial Intelligence at George Mason University. The Center's research is supported in part by the Office of Naval Research under grant No. N00014-91-J-1351 and in part by the National Science Foundation under grant No. IRI-9020266, in part by the Defense Advanced Research Projects Agency under the grant No. N00014-91-J-1854, administered by the Office of Naval Research, and the grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research.

References

Arciszewski, T., Dybala, T. & Wnek, J. (1992). Method for Evaluation of Learning Systems, *Heuristics, The Journal of Knowledge Engineering*, Special Issue on Knowledge Acquisition and Machine Learning, 5.

Bergadano, F., Giordana, A., Saitta, L., DeMarchi, D. & Brancadori, F. (1990). Integrated Learning in Real Domain, *Proceedings of the 7th International Conference on Machine Learning*, (pp. 322–329), Austin, TX.

Bloedorn, E. Michalski, R.S. (1991a). Data Driven Constructive Induction in AQ17-PRE: A Method and Experiments, *Proceedings of the Third International Conference on Tools for AI*, (pp. 9–14) San Jose, CA.

Bloedorn, E. & Michalski, R.S. (1991b). Constructive Induction from Data in AQ17-DCI: Further Experiments, *Reports of the Machine Learning and Inference Laboratory*, MLI 91-12, Center for Artificial Intelligence, George Mason University, Fairfax, VA.

Bratko, I. & Lavrac, N. (Eds.), (1987). *Progress in Machine Learning*, Sigma Press: Wilmslow, England.

Bratko, I. & Kononenko, I. (1986). Learning Diagnostic Rules from Incomplete and Noisy Data. In B. Phelps (Ed.), *Interactions in AI and Statistical Method*, Gower Technical Press.

Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J. (1984). *Classification and Regression Trees*, Wadsworth: Belmont, CA.

Clark, P. & Niblett, T. (1987). Induction in Noisy Domains. In I. Bratko and N. Lavrac, (Eds.), *Progress in Machine Learning*, Sigma Press: Wilmslow, England.

Cestnik, B. & Bratko, I. (1991). On Estimating Probabilities in Tree Pruning, *Proceeding of EWSL 91*, (pp. 138–150) Porto, Portugal.

Cestnik, B. & Karalic, A. (1991). The Estimation of Probabilities in Attribute Selection Measures for Decision Tree Induction, *Proceeding of the European Summer School on Machine Learning*, Priory Corsendonk, Belgium.

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

Michalski, R.S. (1973). AQVAL/1-Computer Implementation of a Variable-Valued Logic System VL1 and Examples of its Application to Pattern Recognition, *Proceeding of the First International Joint Conference on Pattern Recognition*, (pp. 3–17), Washington, DC.

Michalski, R.S. (1978). Designing Extended Entry Decision Tables and Optimal Decision Trees Using Decision Diagrams, *Technical Report No. 898*. Urbana: University of Illinois.

Michalski, R.S. (1983). "A Theory and Methodology of Inductive Learning, *Artificial Intelligence*, 20, 111–116.

Michalski, R.S., Mozetic, I., Hong, J. & Lavrac, N. (1986). The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains, *Proceedings of AAAI-86*, (pp. 1041–1045), Philadelphia, PA.

Michalski, R.S. (1990). Learning Flexible Concepts: Fundamental Ideas and a Method Based on Two-tiered Representation. In Y. Kodratoff and R.S. Michalski (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. III, Morgan Kaufmann: San Mateo, CA, pp. 63–111.

Mingers, J. (1989a). An Empirical Comparison of selection Measures for Decision-Tree Induction, *Machine Learning*, 3, 319–342.

Mingers, J. (1989b). An Empirical Comparison of Pruning Methods for Decision-Tree Induction, *Machine Learning*, 3, 227–243.

Niblett, T. & Bratko, I. (1986). Learning Decision Rules in Noisy Domains, *Proceeding Expert Systems 86*, Cambridge University Press: Brighton, Cambridge.

Quinlan, J.R. (1979). Discovering Rules by Induction from Large Collections of Examples. In D. Michie (Ed.), *Expert Systems in the Microelectronic Age*, Edinburgh University Press.
Quinlan, J.R. (1983). Learning Efficient Classification Procedures and Their Application to Chess End Games. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann: Los Altos, CA.
Quinlan, J.R. (1986). Induction of Decision Trees, *Machine Learning* 1, 81–106.
Quinlan, J.R. (1987). Simplifying Decision Trees, *International Journal of Man-Machine Studies*, 27, 221–234.
Quinlan, J.R. (1989). Documentation and User's Guide for C4.5, unpublished.
Quinlan, J.R. (1990). Probabilistic Decision Trees. In Y. Kodratoff and R.S. Michalski (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. III, Morgan Kaufmann: San Mateo, CA, pp. 63–111.
Smyth, P., Goodman, R.M. & Higgins, C. (1990). A Hybrid Rule-Based/Bayesian Classifier, *Proceedings of ECAI 90*, Stockholm.
Thrun, S.B., Mitchell, T. & Cheng, J. (Eds.) (1991). The MONK's Problems: A Performance Comparison of Different Learning Algorithms, *Technical Report*, Carnegie Mellon University.