# The AQDT-2 USER'S GUIDE
## A Machine Learning Program for Learning Task-oriented Decision Structures from Decision Rules

Ibrahim F. Imam

MLI 96-6

March 1996

# Table of Content

## ABSTRACT

AQDT-2 is a learning system that generates task-oriented decision structures from decision rules. AQDT-2 is the second version of the AQDT leaning system introduced by Imam and Michalski (1993). The system was first designed to learn decision trees from decision rules. Then it was expanded to learn task-oriented decision structures from decision rules or from examples. The system contains many features for representing different decision making situations. These features include the implementation of cost functions that allow the system to optimize the generated decision trees for a given situation. Three attribute selection criteria are used to select the best attribute to be a node in the decision structure, namely disjointness, importance and value distribution of an attribute. This paper includes a description of the input and the output format to the program, a description of the possible parameter settings and a demonstration of how to use AQDT-2 for different decision making situations, and other useful information.

## ACKNOWLEDGMENT

The author would like to thank Eric Bloedorn, Mark Maloof, and Ryszard S. Michalski for their comments, suggestions, and careful review of the paper.

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research is supported in part by the Advanced Research Projects Agency under the grant No. N00014-91-J-1854, administered by the Office of Naval Research, and under the grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, 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.

## 1. INTRODUCTION

A standard approach to determining decision trees is to learn them from a given set of class-labelledexamples. A disadvantage of this approach is that once a decision tree is learned, it is difficult to modify or to learn new decision trees that suit different decision making situations. Such problems arise whenever there is incomplete or unusual reasoning situation. 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. Some attempts has been made to solve such problems using probabilistic or pruning techniques, but due to the inflexibility of the examples, it is difficult to learn a new decision tree for the given situation. 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-2, 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-2 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.

AQDT-2 is implemented in ANSI standard C and is currently available on both SPARC2 stations and PCs.

## 2. THE AQDT-2 ALGORITHM

AQDT-2 constructs a decision tree from decision rules by recursively selecting at each step the "best" attribute according to the attribute ranking measure described in (Imam, and Michalski, 1993), 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.

---

*Input:* A set of decision rules, a decision making situation, and a set of testing examples.

*Output:* A decision structure that suits the given decision making situation.

*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 [**A= i** v ...]. If a branch is assigned values i v j, then associate with it all rules containing condition [**A= i v j** v ...]. 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] ≡ { [x=1] & [y= a] v [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 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 (Bloedorn, et al, 199). 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.

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.*

## 3. AQDT-2 TABLES GUIDES

The input files to AQDT-2 should consists of two tables in addition to the decision rules. These two tables are the parameters table and the attribute table. The parameter tables contains settings of all the learning parameters. These parameters will affect the output decision structure. The variable table should contain information about each attribute including its cost, number of values, type, etc.

### 3.1 Parameters

The mandatory parameters table contains values which control the execution of AQDT-2. AQDT-2 uses different parameters from AQ standard parameters. Here are the set of possible parameters that aqdt can take. AQDT-2 consider the default value for each undefined parameter.

**Mode (Default = dc)**

This parameter indicate the type of rules' cover. The possible values are "dc" for disjoint cover, and "ic" for intersected cover. Usually, it is better to use disjoint rules.

**LEF (Default = 120)**

Specifies the ranking of different attribute selection criteria used by AQDT-2. This parameter should consists of three digits. The first digit represents the order of the disjointness criterion. The second digit represents the order of the attribute importance. The third digit represents wether the value distribution criterion should be used or not. The third digit should be either 0 for "off" or 1 for "on". The difference between the first and the second digits should equal to ±1.

**LEF_tol (Default = 0)**

Specifies the tolerance of the LEF function for the first criterion only. Suppose that the given tolerance is $T$. Consider that $\beta$ and $\partial$ are the evaluation given by the first criterion for attributes $X$ and $Y$. If the absolute difference between ($\beta$ div $T$) and ($\partial$ div $T$) equal to zero, the two attributes are considered equally important for building the decision structures.

**Cost (Default = no)**

Specifies wether to consider the cost of attributes in the process of generating decision structures. This parameter can be assigned to either "yes" or "no".

**Cost_tol (Default = 0)**

This parameter defines the cost tolerance for attributes. Suppose that the given tolerance is $T$. Consider that $\beta$ and $\partial$ are the costs of attributes $X$ and $Y$. If the absolute difference between ($\beta$ div $T$) and ($\partial$ div $T$) equal to zero, then the cost of the two attributes are considered equally important for building the decision structures.

**Test (Default = yes)**

This parameter specifies wether the obtained decision tree should be tested or not. The default setting is "yes", this means there should be a testing file. The format of the input test file can be found in Appendix A.

**Unknown (Default = no)**

In some cases specially when using variable costs, the available information or set of rules is not enough to to drive decisions for some particular cases. Setting this parameter to "no" would result in estimating the best possible decision and assign it to the corresponding leaf node. When this parameter takes value "yes", it generates instead an unknown leaf node. However, another value is suppose to generate a leaf node and associate with it all possible decisions with their probabilities.

**Noise (Default = 10)**

This parameter defines a threshold for the degree of generalization done by the algorithm. Consider that a set of rules $R$ belonging to $n$ decision classes, $C_1$, ..., $C_n$, and associated with one branch $b$ of the tree. Suppose that $\Omega C_i$ is the ratio between the sum of all t-weights of rules belonging to class A and the sum of all t-weights belonging to all other decision classes. If this ratio is less than the given threshold for the other decision classes, AQDT-2 creates a leaf node and assign to it the decision $C_i$. Otherwise, it selects another attribute to further classify the set of rules R.

### 3.2 Variables

The mandatory variables table specifies the names, domains, types, and costs of the variables used to describe events. For more details see the variable tables in Appendix A.

### 3.3 Rules table

Each decision class is described by a set of rules, called rule table. A rules tables have two parts: 1) a rule index and 2) a rule body. The rule body is represented by a conjunction of number of conditions. A conditions can be represented by logical or mathematical expressions.

## REFERENCES

Bloedorn, E., Wnek, J., Michalski, R.S., and Kaufman, K., "AQ17: A Multistrategy Learning System: The Method and User's Guide", Report of Machine Learning and Inference Laboratory, MLI-93-12, Center for AI, George Mason University. 1993.

Imam, I.F. and Michalski, R.S., "Learning Decision Structures from Decision Rules: A method and initial results from a comparative study", in *Journal of Intelligent Information Systems JIIS,* Vol. 2, No. 3, pp. 279-304, Kerschberg, L., Ras, Z., & Zemankova, M. (Eds.), Kluwer Academic Pub., MA, 1993.

Michalski, R.S., and Imam, I.F., "Learning Problem-oriented Decision Structures from Decision Rules: The AQDT-2 System", in The Proceedings of the International Symposium on Methodology for Intelligent Systems, ISMIS-94, Charlotte, NC, October, 1994.

# APPENDIX A. SAMPLE APPLICATION OF AQDT-2

## A.1 Input Format

The input file to AQDT-2 should consists of three sections: 1) the parameters table—contains the parameter settings for AQDT-2; 2) The variable table—contains a list of all attributes used in describing the decision rules, their costs, the number of values per each attribute, and the type of their domains; 3) The decision rules. Figure 1 shows an example of an input file.

```
parameters
mode   lef    cost    cost_tol    lef_tol    test    unknown    noise
 dc    120     no        0           0        yes       no        10

variables
 #    type     levels    cost      name
 1    nom        3       50        x1
 2    nom        3       50        x2
 3    nom        2       50        x3
 4    nom        3       50        x4
 5    nom        4       50        x5
 6    nom        2       50        x6

Pos-outhypo
 #  cpx
 1  [x5 = 1]
    (t : 48, u : 48)
 2  [x1 = 1][x2 = 1]
    (t : 36, u : 36)
 3  [x1 = 2][x2 = 2]
    (t : 36, u : 36)
 4  [x1 = 3][x2 = 3]
    (t : 36, u : 36)

Neg-outhypo
 #  cpx
 1  [x1 = 1][x2 = 2 , 3][x5 = 2 , 3 , 4]
    (t : 24, u : 24)
 2  [x1 = 2][x2 = 1 , 3][x5 = 2 , 3 , 4]
    (t : 24, u : 24)
 3  [x1 = 3][x2 = 1 , 2][x5 = 2 , 3 , 4]
    (t : 24, u : 24)
```

Figure 1: An example of an input file to AQDT-2.

Note that the current version of AQDT is very senstive to spaces and the starting position of the decision rules and their weights. There should be spaces arround the "=" and "," signs. All rules and their weights should start at the same column where the "cpx" sympol starts. The weights should be on sperate line.

In this example, the lef value "120" means that the disjointness criterion was ranked first (according to the first digit) for attribute selection, while the attribute importance ranked second (according to the second digit). In this example, the value distribution criterion was not used (the third digit assigned value zero).

The decision rules describe a discriminant concept (mode=dc). Both the cost function and its tolerance were not used in this case. The LEF tolerance was set to zero. The unknown parameter were set to "no" which means that the system should assign a leaf node with a decision for any case where there is no enough information to continue the classification process.

Finally, the noise parameter refer to the degree of generalization needed during the process of building the decision structure/tree. To explain the meaning of this parameter, let us assume that at a certain node we have a set of decision rules that belong to m decision class. Let us assume that Si is the strength of the decision class Ci at this node. This strength Si is determined as the sum of all the t-weights of rules belonging to the decision class Ci and associated with the given node. For any decision class, consider Ri as the ratio between its strength and the strength of all other decision classes at the given node. If for any decision class, this ratio is less than the value associated with the noise parameter, the program will generate a leaf node and will associate to it the decision class with the maximum strength. Otherwise, it select another attribute to classify the decision rules at this node.

The testing data should be stored in a file using the format in Figure 2. The first line should include the attribute names followed by the term "Dec".

```
x1 x2 x3 x4 x5 x6  Dec
1  2  1  2  4  2   Neg
1  2  1  3  2  1   Neg
1  2  2  3  2  1   Neg
1  2  2  3  4  1   Neg
1  3  1  2  2  1   Neg
1  3  1  2  3  1   Neg
1  3  1  3  3  1   Neg
1  3  2  1  2  1   Neg
1  3  2  3  4  1   Neg
2  1  1  1  4  2   Neg
2  1  1  3  4  1   Neg
2  1  2  1  4  1   Neg
2  2  1  1  3  1   Pos
2  2  1  3  2  2   Pos
2  2  1  3  3  2   Pos
2  2  1  3  4  2   Pos
2  2  2  1  3  2   Pos
2  2  2  1  4  1   Pos
2  2  2  1  4  2   Pos
2  2  2  2  3  1   Pos
```

Figure 2: An example of testing file.

Each lines, except the first, should represent one testing examples. The values should be ordered according to the attributes' order in the first line. Each example should be followed by its decision.

### A.2 Output

AQDT-2 produces output in the canonical format used for representing decision trees. In this formats, each line represents a branch of the decision tree. A branch connects two nodes. If a line contains an attribute assigned to a value, and a decision class, then it connects a node containing that attribute with a leaf node containing the decision class. If a line contains only an attribute assigned to a value, then the line represents a branch from a node to another node. The depth of a node in the tree is measured by the number of pars (e.g., "|") before the node. The depth of the root of the tree is zero. For example, consider the following sub-tree:

```
x5= 1 :   Neg
x5 = 2:
|     x1 = 1:
|     |     x2 = 1 :   Neg
|     |     x2 = 2 :   Pos
|     |     x2 = 3 :   Pos
|     x1 = 2:
```

The root of the tree is x5. In this subtree, x5 has two childern. One is a leaf node assigned with value 1. The other is a node assigned the attribute x1.

In this subtree, four lines connect nodes with decisions, three lines connect nodes with other nodes. The node containing x2 has a parent x1.

When AQDT ran on the data in Figure 1, it produced the decision tree in Figure 3.

```
x5 = 1 :   Neg
x5 = 2 :
|      x1 = 1:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Pos
|      x1 = 2:
|      |      x2 = 1 :   Pos
|      |      x2 = 2 :   Neg
|      |      x2 = 3 :   Pos
|      x1 = 3:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Neg
x5 = 3:
|      x1 = 1:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Pos
|      x1 = 2:
|      |      x2 = 1 :   Pos
|      |      x2 = 2 :   Neg
|      |      x2 = 3 :   Pos
|      x1 = 3:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Neg
x5 = 4:
|      x1 = 1:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Pos
|      x1 = 2:
|      |      x2 = 1 :   Pos
|      |      x2 = 2 :   Neg
|      |      x2 = 3 :   Pos
|      x1 = 3:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Neg
```

Figure 3: A decision tree learned by AQDT from the data in Figure 1.

## APPENDIX B: LEARNING TASK-ORIENTED DECISION STRUCTURES/ TREES

### B.1. Changing Attribute Costs

parameters

| mode | lef | cost | cost_tol | ... |
|---|---|---|---|---|
| dc | 120 | yes | 0 | ... |

variables

| # | type | levels | cost | name | |
|---|---|---|---|---|---|
| 1 | nom | 3 | 40 | x1 | <== changed |
| 2 | nom | 3 | 40 | x2 | <== changed |
| 3 | nom | 2 | 50 | x3 | |
| 4 | nom | 3 | 50 | x4 | |
| 5 | nom | 4 | 50 | x5 | |
| 6 | nom | 2 | 50 | x6 | |

Figure 4: Changing the attribute cost of x1 and x2.

```
x5 = 1 :   Neg
x5 = 2 :
|      x1 = 1:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Pos
|      x1 = 2:
|      |      x2 = 1 :   Pos
|      |      x2 = 2 :   Neg
|      |      x2 = 3 :   Pos
|      x1 = 3:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Neg
x5 = 3:
|      x1 = 1:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Pos
|      x1 = 2:
|      |      x2 = 1 :   Pos
|      |      x2 = 2 :   Neg
|      |      x2 = 3 :   Pos
|      x1 = 3:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Neg
x5 = 4:
|      x1 = 1:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Pos
|      x1 = 2:
|      |      x2 = 1 :   Pos
|      |      x2 = 2 :   Neg
|      |      x2 = 3 :   Pos
|      x1 = 3:
|      |      x2 = 1 :   Neg
|      |      x2 = 2 :   Pos
|      |      x2 = 3 :   Neg
```

Figure 5: A decision tree learned with different costs of x1 and x2.