AQVAL/1--COMPUTER IMPLEMENTATION OF A VARIABLE-VALUED LOGIC SYSTEM VL1 AND EXAMPLES OF ITS APPLICATION TO PATTERN RECOGNITION

by

R. S. Michalski

FIRST INTERNATIONAL JOINT CONFERENCE ON
PATTERN RECOGNITION

October 30 - November 1, 1973
Washington, D. C.

73 CHO 821-9C
AGVAL/1—COMPUTER IMPLEMENTATION OF A VARIABLE-VALUED LOGIC SYSTEM VLV. AND EXAMPLES OF ITS APPLICATION TO PATTERN RECOGNITION

R. S. Michalski
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Abstract

AGVAL/1 is a PL/1 program which synthesizes quasi-minimal formulas of the variable-value logic system VLV. under cost (or complexity) functionals designed by a user from a set of available criteria. The paper presents a brief functional description of AGVAL/1 and reports current results of its application to two pattern recognition problems:

1. The synthesis of a classification rule for distinguishing between patients with cancer of the pancreas, primary cancer of the liver (primary hepatoma) and normal patients.

2. The design of special filters for non-uniform texture discrimination.

1. Introduction

The concept of a variable-valued logic system (VL system) was first introduced at the IFIP Working Conference on Graphic Languages in Vancouver, Canada, May 1972. This concept assumes that every proposition (which conveys certain information or a description of a physical or abstract object) and all the variables in the proposition (used to represent any objects, e.g., other propositions) can assume their own number of 'truth-values' which are selected based on semantic and problem-oriented considerations. This assumption makes the system easily applicable to a variety of practical problems, in particular as a tool for describing objects or classes of objects. The assumption is supported by the observation that humans vary and adopt the 'degree of truth' or 'descriptive precision' of their statements according to the needs of the situation, and do so by changing the 'degree of truth' or 'quantization' of concepts represented by individual words or phrases in the statements. (E.g., when describing a person's height, different degrees of precision may be appropriate in various situations, so we may say that someone is 'tall', or is 'fairly tall' or is '6 feet 2 inches tall'. Each of these phrases implies the use of a different quantization of the concept 'height'.)

From the standpoint of formal logic, traditional binary logic systems and many-valued logic systems are special cases of a VL system.

Paper defined and discussed a particular VL system, called VL1. Important features of the VL1 system are:

1. The formulas of the system (called VL formulas) have very simple interpretation and can be very easily handled and evaluated (especially using parallel-sequential techniques).

2. The system is applicable for expressing and solving a variety of problems of discrete mathematics, especially problems which are intrinsically non-linear or involve variables which are measured on a nominal scale (i.e., when values of the variables are 'numerical names' of independent objects and, therefore, arithmetic relationships between these values have no meaning). The last feature makes the system applicable beyond the area of appliability of established pattern recognition methods, such as statistical techniques (parametric or non-parametric) or discriminant function methods.

The necessary condition for practical application of the system is the computational feasibility of its implementation, which means, in particular, the synthesis and minimization of VL1 formulas.

A computer program, called AGVAL/1, has been developed for the above tasks and the first trial applications of AGVAL/1 to various problems were quite satisfactory.

This paper gives a brief functional description of AGVAL/1 and presents current results from its application to two problems: the synthesis of classification rules for medical diagnosis, and the design of a set of special filters for non-uniform texture discrimination.

2. Definition of the VLV. System

To describe AGVAL/1, knowledge of the VLV. system has to be assumed. Therefore, to make the paper self-contained, we will review briefly the definition of VLV. (as in paper with slight modifications).

The variable-valued logic system VLV. is an ordered quintuple:

$$\{(X, Y, S, R_F, R_X)\}$$

where

- X is a finite non-empty (f.n.) set of input variables

$$x_1, x_2, \ldots, x_n$$

whose domains, called input or independent name sets, are f.n. sets, respectively:

$$H_1, H_2, \ldots, H_n$$

where $$H_i = \{0, 1, 2, \ldots, \mu_i\}$$, $$\mu_i$$--a natural number.

- Y consists of one output variable y, whose domain is a f.n. set, called output or dependent name set:

$$\{0, 1, 2, \ldots, \mu\}$$, \mu--a natural number.

(Consants in H represent 'truth-values' which may be taken by statements (formulas) in the system.)

- S is the set of 13 improper symbols:

$$\{\top, \bot, \land, \lor, \lnot, \lnot, [ ], ( ) \}$$

- $$R_F$$ is a set of formation or syntactic rules which define well-formed formulas (wffs) in the system (VL formulas):

The name AGVAL/1 was derived from 'Algorithm A' applied for the synthesis of Variable-Valued Logic formulas.
1. A primitive constant from H standing alone is a wff.

2. A form \([L \neq R]\) where

\[
\begin{align*}
\# & \in \{z, f, s, k\} & R & \in \{c, v_2\} \\
L & \in \{x_1, v_1\} & x_1 & \in X \\
\_ & \text{a sequence of different non-negative integers separated by ', ' or ': ' and ordered by the relation } < \text{, or a name*} \\
& \text{of such a sequence;} \\
v_1, v_2 & \text{wffs or names** of wffs,} \\
is a wff.
\end{align*}
\]

The form \([L \neq R]\) is called a selector. L and R, left and right parts of the selector, are called the referent (of L) and the reference (of R), respectively. If L is \(x_1\), then \(L\) is called a simple reference. If R is \(c\), then R is called a simple selector. A selector with a simple referent and a simple reference, i.e., a form \([x_1 \neq c]\), is called a simple selector.

A simple reference, \(c\), is called an extended reference, if it contains no ': '. If in an extended reference every maximal (under inclusion) sequence of consecutive integers of length at least three is replaced by a form \(c_1c_2\), where \(c_1\) is the first and \(c_2\) the last element of the sequence, then the obtained reference is called a compressed reference.

3. \(V\), \(-V\), \(V_1 \wedge V_2\) (also written \(V_1 \wedge V_2\)) and 

\(V_1 \vee V_2\), where \(V, V_1, V_2\) - wffs or names of wffs, are wffs.

\(-V\) is called the complement of \(V\)

\(V_1 \wedge V_2\) is called the product (or conjunction) of \(V_1\) and \(V_2\)

\(V_1 \vee V_2\) is called the sum (or disjunction) of \(V_1\) and \(V_2\)

\(R\) is a set of interpretation or semantic rules which assign to any wff \(V\) a value \(v(V) \in H\), depending on values of the variables \(x_1, x_2, \ldots, x_n\).

1. The value \(v(c)\) of a constant \(c \in H\), is \(c\), which is denoted \(v(c) = c\).

2. \(v([L \neq R]) = \begin{cases} \# & \text{if } v(L) \neq v(R) \\ \ast & \text{otherwise} \end{cases}\)

where

\(v(L)\), \(L \in \{x_1, v_1\}\), is value of \(x_1\), if \(L = x_1\); otherwise \(v(v_1)\), i.e., value of \(v_1\)

\(v(R)\), \(R \in \{c, v_2\}\), is the sequence \(c\), if \(R = c\); otherwise \(v(v_2)\), i.e., value of \(v_2\)

\(v([L \neq R]) \neq v(\_) = \begin{cases} \geq & \text{true if } v(L) \text{ is (is not) between any pair of integers in } \_ \text{ separated by '}, ' \\ \leq & \text{true if } v(L) \text{ is smaller than or equal to (greater than or equal to) every integer in } \_ \text{ (in normal use of these relations } \_ \text{ will consist of just one element)} \end{cases}\)

If \(v(L) \neq v(R)\), then the selector \([L \neq R]\) is said to be satisfied.

3. \(v((V)) = v(V)\)

\(v(-V) = \# - v(V)\)

\(v(v_1 \wedge v_2) = \min(v(v_1), v(v_2))\)

\(v(v_1 \vee v_2) = \max(v(v_1), v(v_2))\)

In the evaluation of a \(VL_1\) formula, \(\land\) has higher priority than \(\lor\).

The following is an example of a \(VL_1\) formula and its interpretation:

\[4(x_1 = 0; 7) \land [x_2 \neq 0, 5] \lor 2(x_3 \geq 3) \lor 1(x_5 = 0; 4) \]

The formula (2) is assigned the value 4 (briefly, has value \(4\)) if \(x_1\) accepts value between (inclusively) 0 and 4, or value 7; and \(x_2\) accepts value which is neither 0 nor 5. The formula has value 2 if the previous conditions do not hold and \(x_3\) has value greater or equal to 3. The formula has value 1 if both of the previous conditions do not hold and \(x_5\) accepts value between 0 and 4. If none of the above conditions hold, the formula has value 0.

In the sequel we will consider only a certain kind of \(VL_1\) formulas, called distinguished simple \(VL_1\) formulas (DVL\(_1\) formulas). A DVL\(_2\) is a \(VL_1\) formula which is a sum of simple terms, where each simple term is a product of a constant from \(H\) and one or more simple selectors. If \(x_1\) in a DVL\(_1\), the references of simple selectors are in the form \(c_1c_2\) or \(c\), then it is called an interval DVL\(_1\) formula.

3. Event Space and Cost Functionals for \(VL_1\) Formulas

The interpretation rules interpret \(VL_1\) formulas as expressions of a function:

\[\Sigma: H_1 \times H_2 \times \ldots \times H_n \rightarrow H\]

The set \(H_1 \times H_2 \times \ldots \times H_n\), \(H_j = \{0, 1, \ldots, m_j\}, j = 1, 2, \ldots, n\), includes all possible sequences of values of input variables and is called the universe of events or the event space. The event space is denoted by \(E(h_1, h_2, \ldots, h_n)\) or, briefly, by \(E\), where

\(h_1 = c(h_1) = \#_{i=1}^{m_1}\) (i.e., \(h_1 = h_1 - 1\), which is more mnemonic). The elements of \(E\), vectors \((x_1, x_2, \ldots, x_n)\), where \(x_1\) is a value of the variable \(x_1\), \(x_2 \in H_2\), are called events and denoted by \(e_j\), \(j = 0, 1, 2, \ldots, M\), where \(M = \prod_{i=1}^{n} m_i\). Thus, we can write:

\[E = \{e_0, e_1, \ldots, e_M\}\]

It is assumed that values of the index \(j\) are given by a one-to-one function: \(\gamma: E \rightarrow \{0, 1, \ldots, M\}\), specified by the expression:

\[\gamma(e) = x_n + \sum_{k=n-1}^{0} x_k \prod_{i=1}^{k} m_i\]

\(*\)

\(c(S)\), where \(S\) is a set, denotes the cardinality of \(S\).
\( \gamma(e) \) is called the number of the event \( e \). For example, the number of the event \( e = (2,3,1,6) \) in the space \( E(5,4,2,5) \) is: \( \gamma(5) = 4 + 1.5 + 3 - 2.9 + 2.4 - 2.5 = 119. \)

Assuming that the domains of variables in the formula (2) have cardinalities: \( c(H_1) = 8, c(H_2) = 6, c(H_3) = 7, c(H_4) = 2, c(H_5) = 5 \) and the cardinality of \( H, c(H) = 5 \), the formula is interpreted as an expression of a function:

\[
f : \mathbb{E} = \{0,1,2,3,4\} \rightarrow (0,1,2,3,4)
\]

(6)

In practice we usually deal with functions which may have an unspecified value for some events, that is, with functions:

\[
f^* : \mathbb{E} \rightarrow \mathbb{F} \cup \{\ast\}
\]

(7)

where \( \ast \) denotes an unspecified value ('don't care').

An incompletely specified function \( f^* \) can be considered equivalent to a set \( \{f_1\} \) of completely specified functions \( f_1 \), each determined by a certain assignment of specified values (i.e., values from \( \mathbb{F} \)) to events \( e \) for which \( f^*(e) = \ast \). A \( DLV_1 \) formula which expresses any of these functions \( f_1 \) will be accepted as an expression for \( f^* \).

Functions of the type (7) will be called variable-valued logic functions (VL functions).

In general, there can be a very large number of \( DLV_1 \) formulas which express a given VL function (VL expressions of \( f \)). Therefore, a problem arises of how to construct a formula which is minimal under an assumed cost (or complexity) functional. Depending on the application, different properties of \( DLV_1 \) formulas may be desirable when expressing a given VL function, e.g., the minimal number of terms, of selectors, the minimal 'cost' of evaluation, etc. Thus, it is desirable that in a program which synthesizes minimal \( DLV_1 \) formulas, there be available a number of cost functionals, from which the user may select the most appropriate for his type of application. For computational feasibility and convenience, it is desirable to assume, however, that the available functionals be expressed in one standard form.

In the program AQVAL1 (see Chapter 5) which synthesizes and minimizes disjunctive simple \( DLV_1 \) formulas (\( DLV_1 \) formulas), a functional \( A \) measuring their minimality is assumed to be in the form:

\[
A = \langle a \text{-list}, \tau \text{-list} \rangle
\]

(8)

where

- \( a \text{-list} \), called attribute list (or criteria list), is a vector \( a = (a_1, a_2, \ldots, a_{n_1}) \), where the \( a_i \) denote single- or many-valued attributes used to characterize \( DLV_1 \) formulas.
- \( \tau \text{-list} \), called tolerance list, is a vector \( \tau = (\tau_1, \tau_2, \ldots, \tau_{n_2}) \), where \( 0 \leq \tau_i \leq 1 \), \( i = 2, \ldots, l \), and the \( \tau_i \) are called tolerances for attributes \( a_i \).

A \( DLV_1 \) formula \( V \) is said to be a minimal \( DLV_1 \) expression for \( f \) under functional \( A \) if:

\[
A(V) \equiv A(V_j)
\]

(9)

where

\[
A(V) = (a_1(V), a_2(V), \ldots, a_l(V))
\]

\[
A(V_j) = (a_1(V_j), a_2(V_j), \ldots, a_l(V_j))
\]

\( a_i(V), a_i(V_j) \) denote the value of the attribute \( a_i \) for \( V \) and \( V_j \), respectively. \( V_j, j = 1, 2, 3, \ldots \) are all irredundant \( DLV_1 \) expressions for \( f \) (a \( DLV_1 \) expression is called irredundant if removing any term or selector makes it no longer an expression for \( f \)).

\( \tilde{k} \) denotes a relation, called the lexicographic order with tolerance \( \tau \), defined as:

\[
A(V) \tilde{k} A(V_j) \text{ if } \begin{cases} a_1(V_j) = a_1(V) \text{ or } a_1(V_j) = a_1(V) + \tau \text{ and } a_2(V_j) = a_2(V) + \tau \text{ or } \\ a_2(V_j) = a_2(V) + \tau \text{ or } a_2(V_j) = a_2(V) + \tau \text{ or } \vdots \end{cases}
\]

(10)

Note that if \( \tau = (0,0, \ldots, 0) \) then \( \tilde{k} \) denotes the lexicographic order in the usual sense. In this case, \( A \) will be specified just as \( A = \langle a \text{-list} \rangle \).

To specify a functional \( A \) one selects a set of tolerances, puts them in the desirable order in the \( a \text{-list} \), and sets values for tolerances in the \( \tau \text{-list} \).

4. Synthesis of \( DLV_1 \) Formulas

4.1 Basic Concepts from Covering Theory

The synthesis of a minimal \( DLV_1 \) formula for expressing a given function \( f \) under a functional \( A \) is based on the results of covering theory.\(^{23,34,45,56}\) This theory deals with the problems of expressing sets as unions of certain standard subsets ('building blocks') called complexes. One of the basic concepts is that of a cover of a set against another set.

Let \( E^1 \) and \( E^0 \) be disjoint event sets in a universe \( E \) and sets

\[
L = \bigcap_{i \in I} X_i^{A_i}
\]

(10)

where

\[
A_i \subseteq H_i, I \subseteq \{1,2,\ldots,n\}
\]

\[
X_i^{A_i} = \{e \mid (x_1, x_2, \ldots, x_i, \ldots, x_n) / x_i \in A_i\}
\]

(12)

'building blocks' which are used to express event sets. Event sets \( L \) are called cartesian complexes and sets

\[ X_i^{A_i} \] -- cartesian literals.

A cartesian cover \( C(E^{1}, E^{0}) \) of set \( E^{1} \) against \( E^{0} \) is a set \( \{R_i\} \) of cartesian complexes \( L \) such that

\[
E^{1} \subseteq \bigcup_{i \in I} R_i \subseteq E^{0}
\]

(13)

If sets \( A_i \) are restricted to sequences of consecutive numbers from \( H_i \), then the complexes \( L \) are called
Interval complexes and denoted as
\[ \bigcup_{i=1}^{n} |X_1^i| \]  \tag{14}

where \( n_{X_1^i} = \{e = (x_1, x_2, \ldots, x_n) | x_1 \leq x_2 \leq \ldots \leq x_n \} \). A cover which consists of only interval complexes is called an interval cover. (In general, there can be many different types of complexes, e.g., complexes in which some of the literals are cartesian and some interval, the so-called cyclic complexes, etc.)

It is easy to see that a cartesian cover \( C(E^1|E^0) \) (or an interval cover \( I(E^1|E^0) \)) is equivalent to a \( V^1 \) formula which expresses a function \( f: E = \{0, 1, \ast\} \), assuming that \( E^1 = \{e | f(e) = 1\} \) and \( E^0 = \{e | f(e) = 0\} \). For example, if
\[ C(E^1|E^0) = X_1 \cup X_2 \cup X_3 \cup X_4 \text{ and } I(E^1|E^0) = X_1^2 \cup X_2^3 \]
then the corresponding \( V^1 \) formulas are:
\[ V_C = [x_1 = 1, 3] [x_2 = 0, 2, 3] \cup [x_2 = 1, 2, 4] \text{ and } V_I = [x_1 = 0, 2] \cup [x_2 = 1, 3] [x_4 = 0] \]

4.2 Synthesis of a Minimal \( V^1 \) Formula under a Functional \( A \)

A \( V^1 \) function \( f \) can be specified by a family of event sets:
\[ \{F^0, F^1, \ldots, F^0\} \]  \tag{15}

where \( F^k = \{e | f(e) = k\}, k = 0, 1, \ldots, 0 \). Let \( (E_1, E_2) \) be a minimal cartesian cover of \( E_1 \) against \( E_2 \) under functional \( A \), defined as a cover such that the \( V^1 \) formula corresponding to it is minimal under \( A \).

The synthesis of a minimal \( V^1 \) formula under \( A \) consists of the following steps:
1. Determine the minimal cartesian covers under \( A \):
   \[ N = M(F^1) [F^1 = \cup F^2 = \cup \ldots = \cup F^0] \]
   \[ F^1 = M(F^2 = \cup F^2 = \cup \ldots = \cup F^0) \]
   \[ F^0 = M(F^1 = \cup F^1 = \cup \ldots = \cup F^0) \]
2. Determine the products of selectors which are equivalent to complexes in the covers \( N = N^1, N^0 \).
3. Multiply the products equivalent to complexes in \( N^1 \) by \( f \), in \( N^0 \) by \( \bar{f} \), and so on, in \( N^1 \) by \( \bar{f} \) by \( N^0 \) by \( f \).
4. The union of all terms thus obtained is a minimal \( V^1 \) formula under \( A \) for the function \( f \).

Thus, the synthesis of a minimal \( V^1 \) is reduced to the iterative synthesis of minimal cartesian covers. The synthesis of minimal covers is, however, computationally feasible only in rather simple cases. In general, such synthesis can require the enumeration of an extensive set of possibilities (the proof of this is similar to the proof of Zhuravlev's necessity of enumeration in the minimization of switching functions).

But this set of possibilities (after applying all possible reduction techniques) is usually too large to make the enumeration feasible. Therefore, the only realistic approach is to seek 'good' approximate solutions (except for simple, academic problems).

For the synthesis of cartesian covers the \( A^3 \) ("quasi-minimal algorithm") can be applied. This algorithm, based on the principle of disjoint stars, is a very simple and efficient method for the solution (generally sub-optimal) of any covering problem, it produces the so-called quasi-minimal cover, which is either minimal or approximately minimal under a given cost-functional. In the case when computations do not exceed the assumed memory and time restrictions, \( A^3 \) produces also an estimate \( \Delta \) of the maximal number of complexes in which the obtained and minimal covers can differ
\[ c(N^0) - c(N) \leq \Delta \]  \tag{16}

where \( N^0 \) and \( N \) denote the quasi-minimal and a minimal cover, respectively.

5. AQUAL/I

AQUAL/I is a PL/I program which applies algorithm \( A^3 \) to the synthesis of minimal \( V^1 \) formulas under a functional \( A \). For the lack of space, we will not describe here the construction of AQUAL/I, but will give only its basic functional characteristics.

5.1 Specification of a \( V^1 \) Function

The \( V^1 \) function \( f \), whose \( V^1 \) expression is to be minimized under a functional \( A \), can be specified in the input data of the program in one of three forms:
1. By event sets:
   \[ F^e, F^1, \ldots, F^0 \]
   
   where \( F^e = \{e = (x_1, x_2, \ldots, x_n) | f(e) = k\} \)
   \[ k = 0, 1, \ldots, 0 \]
2. By sets similar to those in 1, but with events specified not as sequences of input variables, but by their numbers (5), i.e., values \( \gamma(e) \):
   \[ F^\gamma = \{\gamma(e) | f(e) = k\} \]
   \[ k = 0, 1, \ldots, 0 \]
3. By a \( V^1 \) formula (which can be already partially minimized; AQUAL/I will try to minimize it further, if possible, with regard to Functional \( A \)).

5.2 Specification of the Minimality Functional \( f \)

A user designs a functional \( A = \langle n\text{-list}, t\text{-list} \rangle \) by selecting a set of attributes from the available attributes, placing them in a desired order in the \( n\text{-list} \), and setting up the \( t\text{-list} \). In the current version of the program the following seven attributes are available:
1. \( t(V) - \) the number of terms in \( V \),
2. \( s(V) - \) the number of selectors in \( V \),
3. \( z(V) - \) the cost of \( V \) specified as \( \sum_{i=1}^{n} z(x_i) \), where \( z(x_i) \) is the cost, specified in the input data of
determining the value of variable \( x_i \), and \( I \) is the
set of indices of these variables (among \( x_1, x_2, \ldots, x_n \),
specified in the input data) which actually
appear in the output \( \mathcal{L}_k \) formula,

4. \( g(V) \) - the 'degree of generalization' defined as:

\[
g(V) = \frac{1}{e_0} \sum_{k=1}^{d} \sum_{i=1}^{n} g(I_{ki})
\]

where

\[
ge(I_{ki}) = \frac{c(I_{ki})}{c(I_{ki} \cap r^k)}
\]

\( I_{ki} \) --the set of events which satisfy the \( i \)-th term
in the sequence of terms in \( V \) with the constant \( k \),

\( s_k \) --number of terms with the constant \( k \)

\( e_0 \) --total number of terms in \( V \)

\( r^k \) = \{ e | r(e) = k \}

5. \( o(V) \) - the ordering criterion (not single-valued
attribute). This criterion can be used when events
in each set \( r^k, k = 1, 2, \ldots, n \), are assigned weight
\( e(e) \) (which represents, e.g., the frequency of event
occurrence or 'importance' of the event) and a \( \mathcal{L}_k \) formula is desirable, in which the first term
\( s_1^k \) of each sequence of terms (\( s_1^k, s_2^k, \ldots \)) with
constant \( k \), \( k = 1, 2, \ldots, n \) will have maximal
possible weight \( e(s_1^k) \), then the next term \( s_2^k \) will
have the maximal possible weight, etc., where:

\[
w(s_1^k) = \sum_{e \in E_k^1} w(e)
\]

and \( E_k^1 \) - the set of events from \( r^k \), which satisfy
\( s_1^k \), but do not satisfy \( s_2^k, s_3^k, \ldots, s_n^k \).

Formally, \( o(V) \) = \( w(T_1), w(T_2), \ldots \) and minimum
\( o(V) \) means that

\[
o(V) < o(V_1)
\]

where \( V_1 \) --all other \( \mathcal{L}_k \) formulas expressing \( f \)
in the lexicographic order

6. \( l(V) \) - the total length of references:

\[
l(V) = \sum_{j=1}^{t} l(T_j)
\]

where

\[
l(T_j) = \sum_{i \in I_j} l(x_i)
\]

\( l_{ij} \)--the index set specifying variables occurring in
\( j \)-th selectors of the term \( T_j \)

\( t(x_i) \)--the number of constants in the compressed
reference of the variable \( x_i \), i.e.

\( t \)--the number of terms in \( V \)

7. \( r(V) \) - relative scope of references:

\[
r(V) = \sum_{j=1}^{t} r(T_j)
\]

where

\[
r(T_j) = \sum_{i \in I_j} \left( \frac{c_{\max} - c_{\min} + 1}{c_{\min} c_{\max}} \right)^2
\]

\( c_{\min}, c_{\max} \)--minimal and maximal value in the
reference of the variable \( x_i \),

\( l_e(x_i) \)--the number of constants in the extended
form of reference of variable \( x_i \),

\( t, I_0 \)--defined as previously.

5.3 Type of Variables

A simple selector \( [x_i \in c], \# \in \{=, \neq\} \), whose
reference \( e \) is \( c_1 c_2 \) or \( c_1 \), is called cyclic selector.
A cyclic selector in which \( \# \) is just '=' is called an
interval selector.

If we want the \( \mathcal{L}_k \) formula to be synthesized to
include only cyclic (interval) selectors for a given
variable \( x_i \), then \( x_i \) is called (in the program) a
cyclic (interval) variable, otherwise a cartesian
variable.

\( \mathcal{A} \text{QUAL/1} \) permits a user to specify the type (cyclic,
interval, or cartesian) of each input variable
independently. Some guidelines for the specification of
the type of the input variables follow.

interval (and secondarily, cyclic) selectors are
usually the simplest for evaluation (depending, in
general, on the way the selectors are represented and
how the formula is evaluated--by a computer program
or by a specialized machine). Also, the synthesis of
minimal \( \mathcal{L}_k \) formulas usually takes considerably less
computational time and memory if variables are specified
as interval (or secondarily, cyclic) rather than as
cartesian variables. On the other hand, \( \mathcal{L}_k \) formulas
with interval (or cyclic) selectors will usually have
considerably more term s and selectors than formulas
obtained when no restrictions on the variables were
assumed.

If a variable takes values which have natural
linear order (e.g., represent temperature, height,
length, grey-level of a picture element, etc.) then it
is usually advisable to specify it as an interval (or
cyclic) variable. If this is not the case (e.g., the
variable represents a set of independent objects,
relations, properties, etc., i.e. is measured on a
nominal scale), then the variable should be specified
as a cartesian variable.

5.4 Restriction Parameters \(<n_s,c_s>\)

Parameters \(<n_s,c_s>\), called max-star and cut-star,
are 'tree pruning' parameters which are used to control
the number of operations, and consequently, the execution time and memory requirements of the program. If a given problem is 'difficult' (i.e., \( n, h, k = 1, 2, \ldots, h, c(k), k = 1, 2, \ldots, h \) are large), then \(<m_1, c_1>\) are set to small natural numbers (e.g., \(<5, 5>, <5, 1>, \ldots\)), which causes a faster execution of the program and decreases memory requirements. The price for this is that the resulting \(V_1\) formula may be 'worse', i.e., further from the minimum with regard to the specified functional A.

A general explanation of the function of these parameters is as follows. The synthesis of a minimal \(V_1\) formula consists of a sequence of operations, each being a multistep process. In every step of the process, one set of complexes is transferred into another, usually larger. If the new set has more than \(m_1\) elements, then it is reduced ('cut') to a smaller set with only \(m_1\) elements, which are the 'best' candidates with regard to the functional A. The set is then the input for the next step of the process in which the parameters \(<m_1, c_1>\) are used again in the same way.

5.5 Mode: \(V_1, D_C, D_C\)

AQUAL/1 permits a user to select among three kinds of \(V_1\) formulas to be synthesized:

'\(V_1\)' - the formula is to be a \(D_1\) formula without any restrictions.

'D_C' - the formula is to be a \(D_1\) formula with the property that the product of any two terms with different constants is 0 ('disjoint covers').

'IC' - the formula is to be a \(V_1\) formula with the property that the product of any two terms with different constants may be non-empty, but must not be satisfied by any event from sets \(F, F_1, F_2, \ldots, F_0\) which are specified in the input data ('intersecting covers').

5.6 Size of the Program

The current version of the program (AQUAL-7) consists of about 600 PL/I statements and the object module produced by the PL/I (F) compiler requires memory capacity of about 60 k bytes (together with the PL/I library routines—about 90 k bytes).

6. Examples of AQUAL/1 Application

We describe here some current results from the application of AQUAL/1 to two specific problems: (1) a medical diagnosis problem, and (2) a problem of non-uniform texture discrimination.

6.1 A Medical Diagnosis Problem

Medical diagnosis is concerned basically with determining a relationship between diseases and available information about patients, in particular, outcomes of medical tests. If the information is properly quantized, then the \(V_1\) system provides a way for attacking this problem. Namely, the system can be applied to determine the relationship from a set of known instances and express it in the form of \(V_1\) formulas.

The process of achieving this goal consists of two phases:

1. Learning phase

Based on a limited number of cases of the diseases under consideration and, for comparison, cases of 'normal health' (where by 'case' we mean a sequence of medical test results and other information about one patient) a \(V_1\) formula(s) is synthesized using AQUAL/1.

2. Testing phase

The obtained formula(s) is then used to diagnose some new cases of the diseases (and of normal health) in order to estimate its diagnostic performance.

To illustrate a medical diagnosis application of \(V_1\), we will report some results from experiments with data characterizing cases of (1) cancer of the pancreas, (2) primary cancer of the liver (primary hepatoma), and (3) normal health.

The two above diseases are rare in the U.S.A. and are frequently misdiagnosed. The correct diagnosis of cancer of the pancreas was made, before operation or autopsy, in 50% of the cases examined in one study and of primary cancer of the liver, ante mortem, in only 11% of the cases in another study. Since the diseases are rare, there is little available statistical information about them, and various statistical classification techniques (using e.g., Bayesian rule or its modifications) are either not applicable or, if applicable, produce a very high error rate.

The experiments which we performed were of two types:

A. CANCER DETECTION: Determination of a classification rule for detecting patients with cancer, either of the pancreas or of the liver.

B. CANCER DISCRIMINATION: Determination of a classification rule for distinguishing between patients with cancer of the pancreas and patients with primary cancer of the liver.

The original data included values of 62 variables corresponding to general medical tests and parameters: (1-23)—admitting information, (24-43)—blood biochemistry and hematology tests, (44-62)—urinalysis, X-rays, and some other tests. The data, however, were incomplete—for various patients different test outcomes were missing. The complete data, necessary for the learning phase (in the current version of AQUAL/1), were available, in a sufficient number of cases, only for some subsets of variables.

A. CANCER DETECTION

1. Learning phase

In these experiments, we used the following 19 variables:

* The data considered in our example were generously supplied by Dr. M. A. Low of Purdue University (presently with Analytical Services Corporation, McLean, Virginia).
Assuming the cost functional \( A = \langle t, s, z \rangle \) (where \( z(x_i) \), costs associated with variables \( x_i \), were all assumed to be 1) and restriction parameters \( \langle a, c, \sigma \rangle = \langle 5, 5, 5 \rangle \), AVAL/1 produced the following formulas:

1. \( V(c/n) = \langle x_{38} = 2.4 \rangle [x_{42} = 0] [x_{43} \neq 0] V \langle x_{3} = 3.5 \rangle [x_{26} = 3][x_{30} = 2][x_{38} = 3][x_{42} = 1.3][x_{43} = 0] V \langle x_{1} = 2.5 \rangle [x_{30} = 1] [x_{38} = 3] V \langle x_{3} = 2.3 \rangle [x_{26} = 3] \) (25)

The formula comprises 4 terms, which consecutively cover (or are satisfied by) \( (28, 29), (7, 7), (6, 3) \) and \( (2, 2) \) events in the set \( F \), respectively (the first number in each pair denotes the total number of events covered by a given term, and the second-the number of events covered by a given term and not covered by any of the previous terms). The formula depends on only seven variables out of 19. The execution time on the IBM 360/75 was \( 29 \) seconds.

2. \( V(n/c) = \langle x_{38} = 1.3 \rangle [x_{29} = 5][x_{30} = 1][x_{31} = 1, 2] \Lambda \langle x_{30} = 5 \rangle [x_{42} = 3][x_{44} = 0] \) (26)

The formula comprises only one term (which covers all 40 events) and depends in toto on only six variables (out of 19). (Note that with one exception all variables in this formula are different from those used in the formula \( V(c/n) \), which is quite surprising.) The execution time was \( 22 \) seconds.

The fact that \( V(n/c) \) has only one term and is considerably simpler than \( V(c/n) \) (4 terms) seems to confirm the common intuition that cases of 'normal health' are more or less similar, i.e. all parameter values fall simultaneously into certain 'normal' ranges, unlike cancer cases which may be quite different because they may represent different types of cancer. (Note that in an interval \( V(n/c) \) each term corresponds to exactly one multidimensional interval in a subspace of \( E \).)

Translating the \( VL_1 \) values of variables back into real values of tests (expressed in units given in the list of variables), the formula \( V(n/c) \) can be paraphrased:

\[
\langle 3 < x_{25} \leq 6 \rangle [x_{32} = 25][x_{30} = 59][5 < x_{31} \leq 25] \Lambda \langle 5000 < x_{36} \leq 11000 \rangle [x_{45} \leq 1] \) (27)

(In the book, 'normal' ranges of the variables used in this formula are listed as: \( 4 \leq x_{25} \leq 5 \); \( 5 \leq x_{29} \leq 20 \); \( 10 \leq x_{30} \leq 40 \); \( 6 \leq x_{32} \leq 13 \); \( 4000 < x_{36} \leq 11000 \); \( x_{45} = \) a few casts may be present normally', page 164.)

2. Testing phase

The formulas \( V(c/n) \) and \( V(n/c) \) were applied individually and jointly (as a conjugate pair) to classify events in testing sets \( T^0 \) and \( T^1 \).

In individual testing of the formulas we assumed that \( V(c/n) \) produces a correct result when a 'cancer event' \( e \in T^0 \), is covered (i.e. satisfies the formula), or a 'normal event' \( e \in T^1 \), is not covered by the formula; \( V(n/c) \) produces a correct result when the opposite is true, i.e. an \( e \in T^0 \) is covered or an \( e \in T^1 \) is not covered.
The performance of these formulas tested individually is summarized in Table 2a. Note that the total number of testing events was different for each formula (60 for \( V(c/n) \) and 68 for \( V(n/c) \), out of original 70). This is because the table includes only results from testing events for which formulas were completely evaluated (as mentioned before, the testing sets included incompletely specified events for which it is not always possible to evaluate a formula).

It should be noted that to evaluate a given formula it is always sufficient to know only the values of the variables which actually appear in the formula (the number of which is usually much smaller than the total number of variables constituting an event). Moreover, in many cases it is sufficient to know the values of only some of the variables appearing in the formula (e.g. it may be sufficient to know the values of variables occurring in only one term of the formula).

As can be seen from Table 2, the formula \( V(n/c) \) performed better than the formula \( V(c/n) \), though both formulas seem to perform reasonably well (\( V(c/n) \) = 83% (55 out of 68) and 89% (61 out of 68) correct diagnostic decisions for cancer and normal events, respectively; \( V(n/c) \) = 88% (49 out of 56) and 92% (27 out of 29) correct diagnostic decisions for cancer and normal events, respectively).

The 35 correctly classified cancer events were 'distributed into' (covered by) consecutive terms of \( V(c/n) \): (20, 28), (5, 9), (5, 2), (1, 0), where the meaning of these numbers is the same as in the formula description.

Joint testing of two formulas (testing of a conjugate pair of formulas) involves judging the joint performance of two formulas for each testing event. Depending on the individual performance of each of the formulas, the joint performance has been categorized as shown on the left half of Table 1 (detection categories). For a given event, each formula, taken separately, can give a correct or incorrect decision (denoted by 1 and 0, respectively). The joint decision is 'cancer', i.e. the unknown event was classified as a cancer event (otherwise as a normal event).

Thus, if the first formula classified an event as a cancer event and the second as a normal event, the event was classified as a cancer event. If this event was in fact a cancer event, then the joint performance was judged to be 'correct'; otherwise (i.e., if it was a normal event) -- 'incorrect'. The assignment of the other performance categories as shown in Table 1 does not seem to require explicit explanation.

The joint performance of \( V(c/n) \) and \( V(n/c) \) was summarized in Table 2b. The joint performance of these formulas was correct for all 47 cancer testing events (100% correct) and for 60 of the 29 normal events (92% correct). Of the 9 normal events classified as cancer events, 2 were classified correctly, 1 moderately incorrectly and 0 strongly incorrect.


B. CANCER DISCRIMINATION

1. Learning phase

In these experiments we tried to determine a rule, in the form of a \( DVL \) formula, for distinguishing between cancer of the pancreas and primary cancer of the liver, based on 22 general medical tests. (These tests were selected from the original 62 variables for the practical reason of obtaining learning sets of a sufficient size.) The 22 variables (constituting events in the present experiments) included all of the 19 previously used variables (with \( x_{26} \) and \( x_{26} \) quantized differently) plus three additional variables:

Admitting Information
\[ x_1 \] Sex \hspace{1cm} (h_1=2)

Blood Chemistry & Hematology
\[ x_{18} \] Sodium (equiv/l) \hspace{1cm} (h_{18}=1)
\[ x_{27} \] Chlorides (equiv/l) \hspace{1cm} (h_{27}=6)
\[ x_{27} \] Hemoglobin (gm/100 ml) \hspace{1cm} (h_{27}=5)

Urinalysis
\[ x_{44} \] Specific gravity (dimensionless) \hspace{1cm} (h_{44}=5)

The set of 30 cancer of the liver events ('liver events') was partitioned into two sets:
\[ F^t \] learning set of 16 (completely specified) events,
\[ T^t \] testing set of 14 (completely specified) events.

The set of 47 cancer of the pancreas events ('pancreas events') was partitioned into two sets:
\[ F^p \] learning set of 18 (completely specified) events,
\[ T^p \] testing set of 29 (completely specified) events.

AQUAL was applied to find a conjugate pair of formulas \( V(t/p) = V(t/p) \), \( V(p/t) \), where \( V(t/p) \) and \( V(p/t) \) are DVL expressions of a function (24), satisfying conditions:
\[
\{ e/f(e) = 1 \} = F^t \quad \text{and} \quad \{ e/f(e) = 0 \} = F^p \quad \text{for} \quad V(t/p)
\]
\[
\{ e/f(e) = 1 \} = F^p \quad \text{and} \quad \{ e/f(e) = 0 \} = T^t \quad \text{for} \quad V(p/t)
\]

The cost functional was assumed to be, as before, \( A = \{ t, s, z \} \), and the restriction parameters \(<m, e, c>\) were \(<30, 15>\).

The obtained formulas were:

1. \( V(t/p) = [x_{24} = 1] [x_{28} = 1] [x_{25} = 1] [x_{25} = 1] [x_{25} = 1] [x_{28} = 1] [x_{29} = 1] [x_{30} = 1] \) \( [x_{2} = 2] [x_{28} = 1] [x_{29} = 1] [x_{30} = 1] V [x_{26} = 1] \) (28)

The formula comprises 3 terms, which consecutively cover (11,11), (5,4) and (1,1) events, respectively, in set \( T^t \). The formula depends on 9 variables (out of 22). The execution time was 61 seconds (this is a significantly longer time than in the previous examples due to the much larger restriction parameters).
2. \[ V(p/c) = [x_1 = 0][x_2 = 0][x_3 = 0][x_4 = 0][x_5 = 0][x_6 = 2][x_7 = 4][x_8 = 6][x_9 = 8] \\
   [x_{10} = 2][x_{11} = 3][x_{12} = 2][x_{13} = 3][x_{14} = 3][x_{15} = 2][x_{16} = 3][x_{17} = 1][x_{18} = 1][x_{19} = 2] \] (29)

The formula comprises 3 terms, which consecutively cover (9, 3), (5, 3), and (3, 3) events, respectively, in set \( F^P \). The formula depends on 10 variables, \( h \) of which \( x_{24}, x_{26}, x_{29}, x_{31} \) were used in the formula \( V(t/p) \). The execution time was 56 seconds.

2. Testing phase

The formulas \( V(t/p) \) and \( V(p/t) \) were applied individually and jointly to classify events in testing sets \( T^P \) and \( F^P \).

The performance of the formulas in both cases was summarized in table 3. Since the purpose of the formulas was to discriminate between two forms of cancer, the performance categories in joint testing were chosen slightly differently, as shown in the right half of table 1 (discrimination categories).

As can be observed in table 3, the performance of the formulas in individual and, as well, in joint testing was completely unsatisfactory (with the exception of classification of pancreas events by formula \( V(t/p) \), which was 56% correct). Note also that the joint testing gave a very high percentage of indecisions (5%).

There can be few potential reasons explaining the results:

1. the size of learning sets was too small for the program to make a proper generalization,
2. the 22 variables used in the experiment, which are all general medical tests, may not have included any tests which are relevant in a sufficient degree to distinguish between cancer of the pancreas and cancer of the liver, thus ruling out the possibility of obtaining positive results in this experiment,
3. the quantification of variables was not fine enough,
4. the type of generalizations of input data, which AQUAL/1 is able to produce, is not adequate for the particular problem,
5. the medical data which we used had errors (e.g., some cases listed in our data as cancer of the pancreas might have been in fact cases of cancer of the liver, or conversely).

1. With regard to reason 1 (learning sets too small), recall that the learning sets included only 16 (set \( F^P \)) and 18 (set \( F^P' \)) events, which makes it quite probable that they were of insufficient size for making adequate generalization. (To more clearly imagine the problem, note that the program was given only \( 16 \times 18 = 288 \) specified events out of \( \approx 0.5 \times 10^6 \) events constituting the event space.)

2. In view of the previous poor results as compared to the previous quite satisfactory results (10% of cases of cancer (47 cases) were detected), reason 3 (irrelevant variables) seems to be more probable. Namely, in the cancer detection experiment, AQUAL/1 was able to produce good rules also based on relatively small learning sets -- \( 40 + 40 = 80 \) specified events out of \( \approx 2.10^{10} \) (though \( \approx 100 \) times larger, relative to the size of the event space, than in the cancer discrimination experiment). Thus, it seems to imply that the variables used in the experiments were sufficiently relevant for detecting the cancer but were not sufficiently relevant for discriminating between the two cancers.

3. It is natural to expect that differences in general tests, which were used in the experiment, will be much more subtle in distinguishing between two types of cancer than in distinguishing between cancer (either type) and 'normal health'. Therefore, for the cancer discrimination experiment a much finer quantization of variables might have been required. Since essentially the same quantization was used in the cancer detection and the cancer discrimination experiments, reason 3 (quantization of variables not fine enough) is perhaps a very significant factor here (new experiments using a finer quantization of variables are desirable).

4. Reason 4 (inadequate type of generalization) seems unlikely in view of the good results of the first experiment.

5. Reason 5 (incorrect data), of course, would undoubtedly cause bad results, if true.

Finally, it should be noted that the problem of cancer discrimination using general medical tests is not of very significant interest for medicine because there exists a specific test ('alpha-feto protein test') which can easily discriminate between these two cancers (once the presence of cancer has been detected). This importance of the experiments on cancer detection than to those on cancer discrimination, when only general medical tests are used.

6.2 Texture Discrimination

This experiment deals with a problem of describing and discriminating non-uniform textures. AQUAL/1, together with some auxiliary programs, has been applied to a solution of this problem. For lack of space, we will give here only a very brief description of the general idea of the method applied and then illustrate it with an example (various aspects of earlier versions of this method were described in papers \( ^{15} \), \( ^{16} \)).

Suppose there is given a set of texture classes \( \{ T^k \} \), \( k = 0, 1, 2, \ldots, n \), and the problem is to find an operator \( t \), which applied to any texture \( x^k \in T^k \) produces the value \( t(x^k) = k \). We will assume here that the \( T^k \subset T^k \), \( k = 0, 1, \ldots, n \), represent digitized textures, namely \( x \times \beta \) matrices with entries, called picture elements, being positive integers between 0 and some number \( \beta \), inclusively. We will also assume that the scale and position of the pictures of the real textures are 'appropriately' adjusted before digitizing and digitizing operations are applied (for learning and, as well, testing phases).

The operator \( t \) (a 'texture descriptor') is assumed here to be a composite operator:

\[ t = (\phi_1 \circ \phi_2 \circ \ldots \circ \phi_m) \circ d \] (30)
where \( \Phi_{1^i} \in (1,2, \ldots, \mu) \) is called the filtering operator of \( i \)-th iteration. \( \Phi_{1} \) is a composite operator:

\[
\Phi_{1} = a \circ l \circ o \circ p \circ v
\]

in which \( a, l, p \) and \( v \) generally dependent on \( i \), are the following operators:

\( a \) - is an 'averaging operator', specified as \( a_{\alpha_1, v_1} \), \( v_1 \) and \( v_2 \) are divisors of \( \alpha \) and \( \beta \) respectively;
\( \alpha = \alpha_1 \times \alpha_2 \) and \( \beta = \beta_1 \times \beta_2 \); some integers.

\( l \) - is a 'grey-level reduction' operator, specified as \( l_{\bar{\bar{g}}} \), which applied to \( T^k \) produces a new matrix \( T^k_2 \) of the same size, but with entries between 0 and \( \bar{\bar{g}} \), inclusively, rather than between 0 and \( \bar{\bar{g}} \), \( \bar{\bar{g}} \leq \bar{\bar{g}} \).

\( p \) - is an 'event-shape' operator, specified as \( p_{n} \), which applied to \( T^k \) produces a sequence of events \( (x_{1},x_{2}, \ldots,x_{n}) \), \( 0 \leq x_{1} \leq \bar{\bar{g}} \) (the sequence may include some events repeated many times).

The 'event-shape' operator \( p \) (and, as well, the operators \( a \) and \( l \)) should be selected appropriately to the type of textures. Potentially this selection can be done by a computer program; in our experiments it was done with human assistance. In the example to be described, \( p \) was an operator, denoted as \( p_{12} \), whose function can be described as follows. Consider a template:

with cut-off elements ('windows') which are marked \( x_{1}, x_{2}, \ldots, x_{12} \) (assume that these windows are of the 'size' of elements in the matrix \( T^k \)). Suppose that this template is placed on the matrix \( T^k \) (say, in the left top corner) and moved, step by step (where steps are of individual element size) through all possible positions in the matrix. In each position \( (i,j) \) an event \( (x_{1},x_{2}, \ldots, x_{12}) \) is extracted, where \( x_{1}, i=1,2, \ldots, 12 \) is the entry of \( T^k \) 'seen' through the template window marked \( x_{1} \).

The sequence of events obtained at the end of this process, \( E^k \), with the sequence of template positions, \( J^k \), is defined to be the result of applying the operator \( p_{12} \) to \( T^k_2 \): \( \langle E^k, J^k \rangle = p_{12}(T^k_2) \)

\( v \) - is an operator which is a VL/1 formula expressing a function

\[
f: E = \{ (x_{1},x_{2}, \ldots, x_{n}) \} \to (0,1,2, \ldots, \mu)
\]

The result of applying \( v \) to \( \langle E^k, J^k \rangle \) is defined as a matrix \( T^k_2 \) such that its entry in position \( (i,j) \) is the value of the formula \( v \) for the event \( e \) extracted in position \( (i,j) \) of the template.

Let \( T(i) \) denote the result of applying the operator \( \Phi_{1} \) to an unknown texture \( T \) and \( T(1) \) - the result of applying the operator \( \Phi_{1} \) to \( T(1) \), \( i=2,3, \ldots, \mu \).

\( T(i) \) is referred to as the transformed texture \( T \) after iteration \( i \).

Operator \( d \), applied to \( T(\mu) \), selects a sample of \( T(\mu) \) (of a size which is generally determined based on statistical considerations) and gives as the result the most frequently occurring value in this sample (in the case where more than one value occurs with the maximum frequency then other samples from \( T(\mu) \) are tried).

The result of applying \( d \) is taken as the 'numerical name' of the class to which the unknown texture is classified (the recognition decision).

Thus, the problem of describing and recognizing textures from some set of texture classes is considered here as the problem of designing an appropriate operator \( t \). There can be in general, many operators \( t \) which can serve as recognizers (or descriptors) for given texture classes. A functional can be introduced for measuring cost (minimality) of these operators (which may include also the cost of estimated error rate) and the problem arises of constructing an operator \( t \) which minimizes the assumed cost functional.

We will give now a simple example of constructing an operator \( t \) and testing its performance. Consider the two pictures shown in figures 1 and 2 (Picture No. 15 and Picture No. 16) as two texture classes. As elements of these classes, consider various reasonably sized samples of these pictures. A problem is to determine an operator \( t \) which applied to any such sample of either picture gives the numerical name of the texture class from which this sample comes.

Pictures 15 and 16 were quantized into 128 x 128 matrices with \( g = 16 \) (16 grey-levels). Figures 3a and 4a show the result of applying to both pictures the operator \( a_{3,5} \circ I_{2} \) (where \( I_{2} \) produced the binary pictures by thresholding the given matrix at the level of the mean grey-level value computed for the 'learning area'). The pictures in figure 3a and figure 4a were divided into two parts: a learning area - which served as a texture sample used to construct the filtering operators \( \Phi_{1} \) and operator \( t \), and a testing area - which served as a texture sample used to test the performance of the \( \Phi_{1} \) and \( t \).

The experiment consisted of three steps: (a) event extraction step, (b) ADVST/L step, (c) filtering and testing step. The results are summarized in table 1.

The sequences \( E^{0} \) and \( E^{1} \) of events \( (x_{1},x_{2}, \ldots, x_{12}) \), obtained after applying operator \( p_{12} \) to learning areas in figures 3a and 4a, are (in table 4a) represented by three sets of events:

\( E^{0} \) - of events which are in \( E^{0} \) but not in \( E^{1} \),
\( \mathcal{E}^1 \) - of events which are in \( \mathcal{E}^1 \) but not in \( \mathcal{E}^0 \).
\( \mathcal{E}^0 \) - of events which are in both \( \mathcal{E}^0 \) and \( \mathcal{E}^1 \).

Columns \( c(\mathcal{E}^1) \), \( c(\mathcal{E}^0) \) and \( c(\mathcal{E}^0) \) give the length of these sequences. In constructing the filtering operators \( \Phi_1 \), a cutting operator (considered as a part of the operator \( p \)) was applied to the sequences \( \mathcal{E}^1 \) and \( \mathcal{E}^0 \): which:

1. removed from \( \mathcal{E}^1 \) and \( \mathcal{E}^0 \) all vectors which occur in these sequences less than \( r \) times (\( r \) - occurrence threshold),
2. computed for the rest of vectors the parameter \( \lambda \) defined as:

\[
\lambda(e) = \frac{e(\mathcal{E}^1)}{e(\mathcal{E}^1) + e(\mathcal{E}^0)}
\]

where \( e(\mathcal{E}^1) \) and \( e(\mathcal{E}^0) \) - number of times the event \( e \) occurs in \( \mathcal{E}^1 \) and \( \mathcal{E}^0 \), respectively.
3. removed from \( \mathcal{E}^0 \) all events for which \( \lambda > 1b \), and from \( \mathcal{E}^1 \) all events for which \( \lambda < ub \), where \( \lambda \) and \( ab \) are parameters called lambda lower bound and lambda upper bound, respectively. \( 0 < 1b \times ub < 1 \).
4. produced sets \( \mathcal{F}^0 \) and \( \mathcal{F}^1 \) consisting of all distinct events remaining in sequences \( \mathcal{E}^0 \) and \( \mathcal{E}^1 \), respectively (note that due to the operation 3, \( \mathcal{F}^0 \) and \( \mathcal{F}^1 \) are disjoint). Columns \( c(\mathcal{F}^0) \) and \( c(\mathcal{F}^1) \) give the numbers of events in sets \( \mathcal{F}^0 \) and \( \mathcal{F}^1 \).

The experiment consisted of six computer runs (jobs):

1, 2 - in which operators \( T \) were constructed as \( \mathcal{VI}_1 \) formulas \( \mathcal{V}(\mathcal{F}^0/\mathcal{F}^1) \), which assume value 1, if an event \( e \in \mathcal{F}^1 \) and value 0, if \( e \in \mathcal{F}^0 \).

3, 4, 5, 6 - in which operators \( T \) were constructed as \( \mathcal{VI}_1 \) formulas \( \mathcal{V}(\mathcal{F}^1/\mathcal{F}^0) \), which assume value 1, if \( e \in \mathcal{F}^1 \), and value 0, if \( e \in \mathcal{F}^0 \).

In table 1c by \( \mathcal{V}_{12}(\mathcal{F}^0/\mathcal{F}^1) \) is meant \( \mathcal{V}_{12}(\mathcal{F}^0/\mathcal{F}^1) \), i.e., first is applied operator \( \mathcal{V}_1(\mathcal{F}^0/\mathcal{F}^1) \) and then \( \mathcal{V}_2(\mathcal{F}^1/\mathcal{F}^0) \); \( \mathcal{V}_{12}(\mathcal{F}^1/\mathcal{F}^0) \), \( \mathcal{V}_{12}(\mathcal{F}^1/\mathcal{F}^0) \), etc., have analogous meaning.

Parameters \( cl \) and \( ct \) (table 1c) called correctness ratio for learning area and correctness ratio for testing area, respectively, are defined:

\[
cl = \frac{CL}{TT} \quad ct = \frac{CT}{TT}
\]

where \( CL \) and \( CT \) are total numbers of events which \( \mathcal{VI}_1 \) formula classified correctly (i.e., produced for the value 1, if \( e \in \mathcal{F}^1 \) and value 0, if \( e \in \mathcal{F}^0 \)) in learning and testing areas, respectively. \( TL \) and \( TT \) are total numbers of entries in learning and testing areas, respectively.

Figures 3a, b and Figures 4a, c show the transformed textures of 1st and 2nd iteration, i.e., results of applying operators \( \Phi_1 = \Phi_2 \circ \Phi_3 \circ \Phi_4 \), i.e., pictures in figure 15 and figure 16 where \( a, t, p \) and \( v \) are defined in table 1 (jobs 1 and 2). As we can see, the densities of '0s' in figure 3c and '1s' in figure 4c are very large, both in learning and testing areas (they are measured by \( cl \) and \( ct \)). Thus, the operator \( d \) will give the correct recognition decisions even if the samples taken from pictures in figures 3c and 4c are relatively very small (recall that the decision will be correct if: the number of '1s' is greater than the number of '0s' in a sample taken from the testing area in figure 3c or, if the number of '1s' is greater than the number of '0s' in a sample taken from the testing area in figure 4c).

7. Concluding Remarks

Let us briefly summarize the most important properties of the \( \mathcal{VI}_1 \) system and its AQUAL/1 implementation, from the application viewpoint:

1. AQUAL/1 logically infers quasi-minimal \( \mathcal{VI}_1 \) formulas (under an assumed cost-functional) from sets of events representing 'facts' or 'true' statements about any objects or classes of objects. This process is problem-independent; it is based entirely on logical properties of the \( \mathcal{VI}_1 \) system.

2. The \( \mathcal{VI}_1 \) formulas inferred by AQUAL/1 are expressions of a function

\[
f: H_1 \times H_2 \times \ldots \times H_n \rightarrow H
\]

where \( H_i \), \( i = 1, 2, \ldots, n \) and \( H \) are f.n. sets of non-negative integers.

An important property of the \( \mathcal{VI}_1 \) system is that every such function can be expressed by a \( \mathcal{VI}_1 \) formula. Since a large number of problems from different fields, in particular various pattern recognition problems, can be formulated as problems of finding an expression for such a function, AQUAL/1 has a potential for broad application.

3. The system can be especially useful for solving problems which are intrinsically non-linear or involve variables measured on a nominal scale (i.e., values of variables denote independent objects and arithmetical relationships between the values have no meaning; such variables are specified in the program as cartesian variables). This is a very unique feature of the system.

4. The \( \mathcal{VI}_1 \) formulas are very simple for interpretation and evaluation. (When only interval variables are used, they correspond to decision rules involving only hyperplanes perpendicular to individual variable axes.)

5. AQUAL/1 automatically reduces redundant variables.

Acknowledgment

The texture experiments reported in the paper were supported by contract AT(11-1)-2138 with the U.S. Atomic Energy Commission. The author gratefully acknowledges the programming help he received from the following people at the various stages of the development of AQUAL/1 and its earlier versions: Val Tareski (AC-1, 2, 3), Peter Reulofei (AC-4, 5, 6), Celsa Praper and Kurt Hirsch (AQUAL/1 --> V1). The author expresses sincere thanks to his research assistants: Mr. Lee Chapman for help in editing the paper and in the experiments with medical data, and Mr. Tse-Wah Wong for help in the experiments concerning the method of texture discrimination. Thanks go also to Mr. S. N. Jayaramnayuni who provided the event extraction program used in the experiments with textures.
List of References


Figure 15 (p): after applying averaging operator $f_{3,3}$ and thresholding at the level of the mean value $\mu$, and thresholding at the level of the mean value.

Learning area

Testing area

Figure 16 (p): after applying filters $V_1(p^2/p)$ (1st iteration), $\epsilon_1 = 0.174$, $\epsilon_2 = 0.835$

Learning area

Testing area

Figure 17 (p): after applying filters $V_2(p^2/p)$ (2nd iteration), $\epsilon_1 = 0.989$, $\epsilon_2 = 0.998$

Learning area

Testing area

Figure 18 (p): after applying filters $V_3(p^2/p)$ (2nd iteration), $\epsilon_1 = 1.0$, $\epsilon_2 = 0.91$
### Event Extraction Step

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Job Description</th>
<th>Pictures</th>
<th>a</th>
<th>t</th>
<th>p</th>
<th>$c(E_0)$</th>
<th>$c(E_1)$</th>
<th>$c(E_2)$</th>
<th>Cutting Operator</th>
<th>$c(F_0)$</th>
<th>$c(F_1)$</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 iteration for $F^0/F^1$</td>
<td>Pic. 15 '1' — (grid with noise)</td>
<td>a</td>
<td>t</td>
<td>s2</td>
<td>p12</td>
<td>159</td>
<td>129</td>
<td>66</td>
<td>2</td>
<td>&lt;0.25</td>
<td>&gt;0.75</td>
</tr>
<tr>
<td>2</td>
<td>2 iteration for $F^0/F^1$</td>
<td>Pic. 16 '0' — (grid 4)</td>
<td>a</td>
<td>t</td>
<td>s2</td>
<td>p12</td>
<td>111</td>
<td>159</td>
<td>2</td>
<td>2</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>3</td>
<td>1 iteration for $F^0/F^1$</td>
<td></td>
<td>a</td>
<td>t</td>
<td>s2</td>
<td>p12</td>
<td>225</td>
<td>152</td>
<td>68</td>
<td>3</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>2 iteration for $F^0/F^1$</td>
<td></td>
<td>a</td>
<td>t</td>
<td>s2</td>
<td>p12</td>
<td>288</td>
<td>296</td>
<td>15</td>
<td>2</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>5</td>
<td>3 iteration for $F^0/F^1$</td>
<td></td>
<td>a</td>
<td>t</td>
<td>s2</td>
<td>p12</td>
<td>79</td>
<td>170</td>
<td>3</td>
<td>2</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>6</td>
<td>4 iteration for $F^0/F^1$</td>
<td></td>
<td>a</td>
<td>t</td>
<td>s2</td>
<td>p12</td>
<td>17</td>
<td>93</td>
<td>0</td>
<td>2</td>
<td>0.25</td>
<td>0.75</td>
</tr>
</tbody>
</table>

- a = averaging operator
- t = grey-level operator
- p = event-shape operator
- $r$ = occurrence threshold
- $lb$ = lambda lower bound
- $ub$ = lambda upper bound

### AQUAL/1 Step

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Event Space and h</th>
<th>Restriction $&lt;m_o, c_o&gt;$</th>
<th>Cost-Functional $A$</th>
<th>VL Formula</th>
<th># of Terms</th>
<th># of Selectors</th>
<th># of Variables</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$F^{2,2,2,2,2,2,2,2,2,2,2}$</td>
<td>$&lt;1,1&gt;$</td>
<td>$c$-list: (2,3,5) $t$-list: (0.1,0.1,0.1,1)</td>
<td>$V_1(F_0/F^1)$</td>
<td>22</td>
<td>121</td>
<td>12</td>
<td>132</td>
</tr>
<tr>
<td>2</td>
<td>$&lt;$</td>
<td>$&lt;1,1&gt;$</td>
<td>$c$-list: (1,2,3) $t$-list: (0.1,0.1,0.1,0.1)</td>
<td>$V_2(F_0/F^1)$</td>
<td>12</td>
<td>68</td>
<td>12</td>
<td>126</td>
</tr>
<tr>
<td>3</td>
<td>$&lt;$</td>
<td>$&lt;1,1&gt;$</td>
<td>$c$-list: (2,3,5) $t$-list: (0.1,0.1,0.1,0.1)</td>
<td>$V_2(F_0/F^1)$</td>
<td>8</td>
<td>32</td>
<td>12</td>
<td>67</td>
</tr>
<tr>
<td>4</td>
<td>$&lt;$</td>
<td>$&lt;1,1&gt;$</td>
<td>$c$-list: (1,2,3) $t$-list: (0.1,0.1,0.1,0.1)</td>
<td>$V_2(F_0/F^1)$</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>$&lt;$</td>
<td>$&lt;1,1&gt;$</td>
<td>$c$-list: (2,3,5) $t$-list: (0.1,0.1,0.1,0.1)</td>
<td>$V_2(F_0/F^1)$</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

### Filtering and Testing Step

( application of the operator: $a_{1-p-v}$ )

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Operator $v$</th>
<th>Correctness Ratio for Learning Area (cl)</th>
<th>Correctness Ratio for Testing Area (ct)</th>
<th>Correctness Ratio for Learning Area (cl)</th>
<th>Correctness Ratio for Testing Area (ct)</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$V_1(F_0/F^1)$</td>
<td>0.784</td>
<td>0.935</td>
<td>0.244</td>
<td>0.735</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>$V_2(F_0/F^1)$</td>
<td>0.969</td>
<td>0.998</td>
<td>1.0</td>
<td>0.911</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>$V_3(F_0/F^1)$</td>
<td>0.833</td>
<td>0.870</td>
<td>0.918</td>
<td>0.735</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>$V_4(F_0/F^1)$</td>
<td>0.958</td>
<td>0.974</td>
<td>0.946</td>
<td>0.921</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>$V_5(F_0/F^1)$</td>
<td>1.0</td>
<td>0.997</td>
<td>1.0</td>
<td>1.0</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>$V_6(F_0/F^1)$</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>26</td>
</tr>
</tbody>
</table>

### Summary of the experiment on texture discrimination

Table 4.