UIUCDCS-R-77-840

UILU-ENG 77 1724

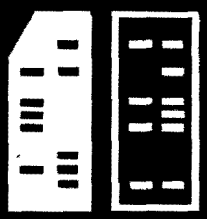
YAQ: A 360 Assembler Version of the Algorithm A^q and Comparison with Other PL/I Programs

77-4

by

Edward Yalow

May 1977



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

YAQ: A 360 Assembler Version of the Algorithm A^q

and Comparison with Other PL/I Programs

by

Edward Yalow

Department of Computer Science
University of Illinois
Urbana, Illinois

May 1977

This work was supported in part by the National Science Foundation, Washington, DC, Grant No. NSF MCS 74-03514.

I. INTRODUCTION

This paper provides a users guide to the use of the program AQVAL/7 version YAQ. In addition a description of YAQ's contribution to the AQVAL/7 algorithm and a comparison with the other versions of AQVAL/7 is given. YAQ is a program that infers a quasi-optimal disjunctive simple formula of the variable valued logic system VL1. In this version a formula distinguishes classes of events and generalizes over unspecified events. The formulas are formed from a description of the event space (number of variables and their domains) and a set of training events. The degree of generalization can be controlled by a "mode" parameter.

II. INPUT

There are eight (8) input parameters which may be specified in a semi-order free format. The general format for a parameter is:

KEYWORD=VALUE

where KEYWORD is the parameter being specified

= specifies that the keyword value follows

and VALUE is one of the allowable values the keyword may take.

Keywords are seperated by one or more blanks.

In addition the following symbols are used in the explanation of the keywords and their values as follows:

ddd	is a one to three digit number
a	is a one to eight character alphameric string beginning with a alphabetic character
?	means that there is no default and a value must be specified

The input that is read is assumed to be 80 character card images with columns 73:80 ignored. Keywords and values may cross card boundaries.

The keywords, their allowable values, and their meanings follow:

KEYWORD	values	meaning
MODE		the type of cover to generate (degree of generalization)
	I	intersecting covers. An unspecified event may be in more than one class. Specified events are covered by one cover only.
	II	same as I but with the covering complexes minimized.
	D	disjoint covers. No events, specified or not, is in more than one class.
	DD	Same as D but with the covering complexes minimized.
	V	VL. VL formulas are generated such that the class covers are order dependent and an event, specified or not is in the first class that covers it. The last class does not have a cover formula but is the default if the event is not covered by a previous class.

VR Same as V but with the covering complexes minimized.

R Reverse VL. VL mode with the classes covered in reverse order.

RR Same as R but with the covering complexes minimized.

EX.: MODE = I

MCL The MAXIMUM number of classes TO BE COVERED. AS THE DEFAULT IS LARGE THIS PARAMETER NEED NOT BE SPECIFIED EXCEPT IN THE RARE CASE THAT A STILL LARGER NUMBER OF CLASSES IS PRESENT.

ddd an integer in the range of 1:255. DEFAULT = 50

MCL = 100

MAXSTAR The maximum size an intemediate star may take before trimming is done. Since trimming contributes a significant amount of the computation done there may be some advantage in using a maxstar significantly larger than cutstar.

ddd an integer in the range of 1:999. DEFAULT = 999

EX.: MAXSTAR = 16

CUTSTAR The size to which an intemediate star is trimmed to.

ddd an integer in the range of 1:999. DEFAULT = 1

EX.: CUTSTAR = 2

PRTMODE The amount of output desired. this is used primarily for debugging purposes.

1 print all events and covers

5 prints covers only. DEFAULT = 5

EX.: PRTMODE = 1

TITLE a The value of title is printed on the top of every page of output after the one it appears on.

EX.: TITLE = ANYTHING

NOVAR the number of variables (dimensions) of the event space.

ddd an integer in the range of 1:255. DEFAULT = ?

EX.: NOVAR = 4

VARSZ This is a set of NOVAR numbers each optionally followed by one or two modifiers enclosed within a set of parenthesis. Each I-th number is the range that the I-th variable may take. If the variable range value is followed by a number within parenthesis, then that number is the weight of the selector. An "I" or a "F" is taken to mean either an interval variable or factor variable respectively. Neither, either, or both weight and type may be specified for each variable range. Blanks are used to seperate the variable specifications. There is a necessary set of parenthesis enclosing the entire varsz specification.

(DDD(DDDbI)...bDDD()) DEFAULT = ?

Ex.: VARSZ=(3 4) no modifications.

Ex.2: VARSZ=(3(5) 4 2(I)) singly modified

Ex.3: VARSZ=(6(1 F) 7(I 2) 6) doubly modified

NOTE: NOVAR must preceed VARSZ.

EVENTS This is the most complicated parameter . Events is the means of specifying the training events to be used. The format is an alphabetic 1 to 8 character classname followed by 0 or more events. Each event may optionally be followed by a weight enclosed in parenthesis. An event is specified either by a gamma notation enclosed in angle brackets or by a set of NOVAR numbers, each within the range specified in VARSZ with blanks used as seperators. A weight is a number between 0 and 255 enclosed in parenthesis. It follows the last variable specification of the event it is modifying. If a class name is repeated then the events following that class name are appended to the events in the first occurence of the class. Thus training events in a single class need not be contiguous. Classes are covered in the order in which the classnames are originally specified. NOTE: Both VARSZ and NOVAR must preceed EVENTS.

classname event (weight) ...event (weight) classname event ...

where: classname is a one to eight character string beginning with an alphabetic character.

event is a set of NOVAR digits each within the range specified in varsz or an event specified in gamma notation and enclosed in angle brackets -"<>" .

A number in gamma notation ranges between 0 and 1 less than the product of the ranges specified in

VARSZ.

(weight) is a number between 0 and 255 enclosed in"()"

Ex.1:

VARSZ=(2 3 4) EVENTS=C1 1 2 3(1) 1 2 2 (2) C2 0 0 0 0 0 1 (4) C1 1 2 1

C1 has three events : (1 2 3) (1 2 2) and (1 2 1)

their weights are 1,2 and 1 respectively.

C2 has two events: (0 0 0) and (0 0 1)

their weights are 1 and 4 respectively.

Ex.2: same as Ex.1 except using gamma notation

VARSZ=(2 3 4) EVENTS=C1 <23>(1) <22>(2) C2 <0> <1>(4) C1<21>

Ex.2: using mixed notation this time.

VARSZ=(2 3 4) EVENTS=C1 <23>(1) 1 2 2(2) C2 0 0 0 <1> (4)C1 <21>

Any card that begins with an asteric ('*') in column one is taken as a comment card. A comment card is echoed on the output but otherwise ignored.

JCL USED:

The simplest way to use YAQ is to use the proc YAQ as follows:

```
//jobname       JOB
/*ID PS=psnumber
/*ID CODE=code
:             :
:             :
addition id cards if necessary
```

```
:           :
:           :
//procstepname EXEC YAQ
//AQ.SYSIN DD 8
control cards as described above
/*
```

The proc also allows one to specify an optional maximum region size for the aq step. the keyword is 'r' and should be placed after the procname as shown in the following example.

```
//PROCSTEPNAME  EXEC YAQ,R=90K
```

The default for R is 100k which should be more than ample for most applications as can be shown in the testing data below. The program YAQ may also be called as a subroutine of either a PL/I program or a 360/assembly language program as follows:

From PL/I:

```
CALL BOOT;  /*calls the bootstrapping proceedure
                    that follows */
 INCLUDE CALLAQ;
/*  callaq loads, calls, and unloads each of the
        two modules comprising YAQ  8/
```

From 360/assembler:

```
LINK  EP=AQMOD  load and execute program to
*                    generate covers
```

```
                LOAD  EP=YPRT   load print program
                CALL  PL1MAIN   execute print program
                DELETE EP=YPRT  unload print program
```

FURTHER EXAMPLES OF INPUT CONTROL PARAMETERS

```
//PROCNAME      EXEC YAQ
//AQ.SYSIN      DD   *
*       EXAMPLE OF MINIMUM AMOUNT OF CONTROL PARAMETERS NECESSARY.
 MODE  = I
 NOVAR = 1
 VARSZ = ( 4  )
 EVENTS =
 CLASSONE
 3
 SECLASS        2        1
 CL3    3
/*
//PROCN~2       EXEC YAQ
//AQ.SYSIN      DD   *
*       EXAMPLE TWO
 OUTSTAR = 2   MAXSTAR = 2     MODE = I        VARSZ  = ( 3 2 )
       EVENTS =         CL1    0 0   0 1       CL2    1 0    1 1
/*
```

ABOVE EXAMPLES WITH EXPLANATORY COMMENTS INTERSPERSED

```
*       EXAMPLE OF MINIMUM AMOUNT OF CONTROL PARAMETERS NECESSARY.
*       NOTE:  THIS AND PREVIOUS LINE ARE COMMENT CARDS.
 MODE = I
* USE INTERSECTING COVERS
 NOVAR = 1
* EACH EVENT CONSISTS OF JUST ONE VARIABLE
 VARSZ = ( 4 )
* THE ONE VARIABLE MAY TAKE ANY OF THE FOUR VALUES 0,1,2, OR 3
 EVENTS =
* THE SPECIFICATION OF CLASSNAMES AND EVENTS FOLLOW
 CLASSONE
* THE NAME OF THE FIRST CLASS IS "CLASSONE"
 3
* THE EVENT HAS ITS VARIABLE EQUAL TO 3
 SECLASS       2       1
* THE SECOND CLASS HAS A CLASS NAME OF "SECLASS" AND TWO EVENTS WHOSE
* VALUES ARE  2 AND 1 RESPECTIVELY
 CL3   0
/*
* THE LAST CLASS HAS ONE EVENT WHOSE VALUE IS 0 AND WHOSE CLASS NAME IS
* "CL3"
*       END OF EXAMPLE
*       EXAMPLE TWO
 MODE = I MAXSTAR = 2  CUTSTAR = 2
* IN THIS EXAMPLE MAXSTAR AND CUTSTAR ARE BOTH SPECIFIED.  WHEN AN
```

```
* INTERMEDIATE STAR EXCEEDS TWO COMPLEXES IN SIZE IT WILL BE TRIMMED
* BACK TO TWO COMPLEXES.  IN THE PREVIOUS EXAMPLE MAXSTAR WAS ALLOWED
* TO DEFAULT TO A VALUE OF 999 AND CUTSTAR TO A VALUE OF 1.  IN EFFECT
* THIS MEANT NO TRIMMING (EXCEPT IN THE UNLIKELY EVENT AN INTERMEDIATE
* STAR DID EXCEED 999 COMPLEXES IN WHICH CASE IT WOULD BE TRIMMED BACK
* TO ONE COMPLEX).  NOTE THAT MORE THAN ONE PARAMETER MAY BE SPECIFIED
* ON ONE LINE.
 VARSZ = ( 3 2 )
* THIS MEANS THAT THE EVENT SPACE IS TAKEN TO BE A 3 X 2 MATRIX.
* VARIABLE ONE TAKES ON VALUES BETWEEN 0 AND 2, VARIABLE TWO BETWEEN
* 0 AND 1.
      EVENTS = CL1   0 0   0 1   CL2  1 0   1 1
* THIS EXAMPLE CONSISTS OF TWO CLASSES EACH CONTAINING TWO EVENTS.
* THE CLASS NAMES ARE "CL1" AND "CL2".  CL1 CONSISTS OF THE TWO EVENTS
* WHOSE FIRST VARIABLE IS 0 AND WHOSE SECOND VARIABLE IS 0 AND 1
* RESPECTIVELY.  CL2 CONSISTS OF THE TWO EVENTS WHOSE FIRST VARIABLE IS
* IS 1 AND WHOSE SECOND VARIABLE IS 0 AND 1 RESPECTIVELY.  NOTE THAT
* THE FORMAT IS FREE.  ALSO NOTE THAT THE CLASS NAMES DELIMIT THE
* EVENTS INTO THE VARIOUS CLASSES.
```

SUMMARY TABLE OF KEYWORDS

KEYWORD	VALUES	DEFAULT
MODE	I,D,V,R	I
NCL	0:255	50

MAXSTAR 1:999 999

CUTSTAR 1:999 1

PRTMODE 1,5 5

TITLE CHAR STRING NULL STRING

NOVAR (LIST OF INTEGERS) NO DEFAULT

EVENTS LIST OF CHAR STRINGS AND INTEGERS NO DEFAULT

WEIGHT (1):(255) follows an event or var range 1

TYPE I,F follows a var range 1

III. OUTPUT OF THE PROGRAM:

AQ INPUT

NOVAR = 4 VARSZ = (4 4 4 4)

* THIS IS AN EXAMPLE OF A COMMENT CARD

MAXSTAR = 24 CUTSTAR = 8 MODE = V

* THIS EXAMPLE IS TAKEN FROM A PAPER BY R.S.MICHALSKI

* EXPLAINING GENERALIZED LOGIC DIAGRAMS

* REF(1) - REPORT #463 UOI pub.

EVENTS =

CLASS1

0 3 3 0 0 3 2 1 1 2 3 0 1 3 3 1

0 2 3 1 1 3 2 1 1 2 2 0 1 2 3 1

3 1 0 3 2 1 0 3 2 0 1 2 3 1 1 2

2 1 1 2 3 1 0 3 2 0 0 2 3 0 1 2

CLASS2

3 0 1 0 1 2 1 2 0 2 0 1 1 1 1 0

```
3 3 3 2          2 3 3 2          2 3 3 0          3 3 2 3

3 0 3 0          2 1 3 1          0 3 1 3          0 2 0 3

3 3 3 0          2 0 3 2          0 1 1 1          1 1 0 1

CLASS3

2 0 1 3           2 0 2 3

THERE ARE 00034 EVENTS     16 BIT LONG COMPLEXES.

 CUR ISTAR   27 MAX ISTAR =  96 CUR STAR =   11 LGST STAR =  25

   MAX STAR = 24

DELTA =        3 TRIM CNT =     1

ELAPSED TIME IS  000.418

 CUR ISTAR   45 MAX ISTAR =  76 CUR STAR =   14 LGST STAR =  22

   MAX STAR = 24

DELTA =        0 TRIM CNT =     0

ELAPSED TIME IS  000.664

 SNAP NUMBER  001 - COVER FOR CLASS -     CLASS2

COVER

C1:   (X1=0:1) (X3=0:1)

C2:   (X1=2:3) (X3=2:3)

 SNAP NUMBER  002 - COVER FOR CLASS -     CLASS3

STAR                    - QUEUE IS EMPTY

COVER

C1:   (X2=0:2) (X3=1,3) (X4=3)

'CLASS2''1100111111102111'B'0011111100111111'B'CLASS3''1111111100101000

1'B
```

IV. PROGRAM IMPLEMENTATION

AOMOD is run in two stages. The first being a pair of load modules: one that reads and interprets the users input and the other that generates the appropriate class covers. This load module is 2100 lines of 360/assembler code consisting of ten modules and an additional 400 lines of macro statements and copy-code. The second stage is a 800 line PL/I load module, YPRT, the print module and is composed of 5 proceedures.

DATA STRUCTURES.

Each event and complex is represented as a biunary string as in [1]. All these (event and complex) biunary strings are further organized into doubly linked lists. The header of each list is a dummy string and the hi-order bit of its pointers are flagged. There is a seperate list for each of the following: variables, events in each class, events in f0, events in f1, extension complexes, intemediate star, star complexes, cover complexes, events covered by previous stars, events covered by the current cover chain.

MACROS.

Approximately a dozen macros were coded to aid in the implementation of AOMOD. The most important of these is the COVER macro. A complex, A, covers a complex or event, B, if A intersect B equals A. Put another way COVER(A,B) is true if AB=A. The actual code follows:

```
        MVC   TEM,A      tem=A
        NC    TEM,B      tem=AB
        CLC   TEM,B      test if covered.
```

Other often used macros are described briefly as follows:

SETUP used for linkage

RTCHN moves an event or complex from one chain to another.
(does a delete and add)

MOVECHN moves a group of events or complexes from one chain to another.

PRFCHN deletes a group of events or complexes from active use.

AQMOD MODULES.

The function of the individual modules is as follows:

AQINI This module reads and interprets the user input, initializes many variables, and gets core for the events and some of the complexes. Once the input has been completely read in AQCOV is called

AQSYSIO This module does the actual reading in of input, echoing of output, and also writes out all error messages.

AQINP This module does some minimal initialization and calls AQINI and AQCOV to initialize and cover the training set.
to generate a cover for each class.

AQCOV After each cover has been generated, AQPRT1 is then called to put the cover in a temporary dataset for later printing. Also a message containing statistical information is printed.

AQEY This module generates the covers for each class (the

uncovered queue) passed to it. It does this by calling STARGN to generate a star for an uncovered event from the class being covered. Events covered by the star just generated are moved from the uncovered queue to the star-covered queue. The best complex is then chosen from the star and the events it covers are now moved to the covered queue. This best complex is added to the cover. Should the uncovered queue be emptied while the star-covered queue is not, then events are taken from the star covered queue and a star is generated. As each star is generated DELTA, a measure of the distance of the cover from the optimal possible, is incremented. Next the best complex from the star is added to the cover and the events it covers are put in the covered queue. When both the uncovered queue and the star-covered queue are empty a cover for the class has been generated and AOPY returns.

AOST This module generates a star for a given event. It does this by ordering the events in EK (f0 - the events against which the cover is to be generated) in ascending distances from the event to which a star is to be generated. Then it generates the extension of the star event against the closest event from EK which is outside the generated intermediate star. (note: the intermediate star is initially null.) This extension is then intersected with the intermediate star thus far generated to form a new intermediate star. If the intermediate star generated at this point has more than maxstar complexes in it, it is trimmed to cutstar elements according to the algorithm described in the TRIM module. When there are no

longer any elements from EK outside the current intermediate star, AQST sets the star equal to the current intermediate star and returns.

AQTR This module trims the current intermediate star down to the best cutstar elements. It does this by calling EVALST to evaluate each complex and then ordering all the elements in ascending sequence. The entire star is then freed and the best cutstar elements are put back into the star. AQTR then returns.

AQEVST This module creates an array of evaluation blocks used by AQTR and AQEY to evaluate complexes in the current star or intermediate star. Each of these blocks consists of a last element flag, a count of events covered, a count of sellectors (varriables) used in the complex, a pointer to the complex being covered, and a value of the complex computed as follows:

VALUE = #eventscovered + NOVAR -#selectors in complex

where NOVAR is the total number of variables in the event space. At the point where VALUE is computed, optionally a user specified routine may be called to return a value in a non-standard way.

AQPRT1 This module takes the contents of the queues specified in the calling routine and writes them to a temporary dataset for later processing and printing. Although used primarily for debugging purposes when dumping all or several event and complex queues, whenever a cover for a class is found, AQPRT1 is called to save the cover for later printing and evaluation.

AQMVCH This module is called for moving a chain of events from one

queue to another. It does this by inserting the first element of the chain to be moved before the target buffer of the queue to which the chain is to be moved. The last element of the chain is like wise chained to the element that was originally before the target buffer. The buffer before the first element of the chain moved is then linked to the buffer after this chain. AQMVCH then returns.

AQRCHN This module deletes a buffer from the chain it is currently in and rechains it after the target buffer pointed to by the calling routine.

AQABND This module is called whenever any of the modules above discovers an internal logic error. (This is usually a bad chain of some sort.) When called AQABND will call AQPRT1 to print out all the chains (i.e. f0-events, uncovered events, star-covered events, covered events, intermediate star, incomplete cover, etc.) as well as a short message describing the error. All open files are closed and an abend exit with a core image dump is taken.

YPRT PROCEDURES

PRINT This is the main proceedure which acts as a caller or monitor for the proceedures described below. Its secondary function is to open and close all datasets.

TITLE This proc reads in a title card and prints out the card image. In addition, if a new class name is discovered it notes it for possible later use by EVALCOV.

VAR This proc reads in the varsz array.

WRITEV This proc reads and write events from a particular queue.

WRITPLY This proc reads and writes complexes from a particular queue. In addition if the complexes are part of a class cover it saves them and reformats them according to whether they will later be used by EVALCOV or CONFUS.

EVALCOV This proc evaluates the cover chains saved by WRITPLY against events in a TEST dataset on the basis of percent correct descisions, false positive descisions, false negative descisions, percent of events undescisive , that is not covered by any class cover and percent of events ambiguous, that is covered by more than one class.

CONFUS This proc (written by James Larson,) evaluates the covers saved by WRITPLY against events in a TEST dataset on the basis of percent correct descisions, count of events covered by each class cover and each complex within that cover, count of events ambiguous, and count of events undescisive.

V. COMPARISON OF 4 VERSIONS OF AQVAL/7

A. FEATURES

Ease of use - This is a subjective evaluation taking into account the number of parameters that must be specified, the ease of updating training events, the amount of control has over the covering process, the amount of control one has over core usage and CPU usage, and the amount of JCL necessary.

Clist - This is a parameter that allows the user to specify a covering order other than that in which the training events aRE

presented.

Negative events - To specify an event as an negative event means that it is known that this event does not belong to a particular class but it is unknown what class this event truly belongs to.

Selector restrictions - This is a means of specifying a dependency of a subset of variables upon another subset of variables. For example if it is impossible for variable X2 to take on any value other than one when variable X3 has value one then any complex containing [X2=1][X3=1] is not allowed to occur in thecover for that class containing the restriction.

Formula updating - Sometime it happens that one already has covers for a set of classes but an addition has been made to the training set. Well, ITCN3 allows one to specify these covers and then adjusts the covers to conform to the new training set. This allows a considerable saving of computation time although the new covers are somewhat further from optimal than would otherwise have been produced.

English output - This is a feature that allows one to specify a translation of the variables and variable values into meaningful english mnemonics. The result of this process is that covers are printed something like "[FLOWER=ROSE][COLOR=RED]" instead of "[X1=2][X2=1]".

Parametor echoing - This feature allows the user to see how he typed in his control parameters. It is printed on the output as

it appeared in the input. Note that all versions will let the user know what values are used for the parameters he specifies.

Event echoing - This feature allows the user to see what events he has specified in the training set. The format of this differs amonst the four versions. YAQ simply echoes the users input printing a potentially compact representation of the events. ITCN3 reformats the events so that there is a maximum of one event per line and prints only the values for each variable. AQ5a and AQ7 in addition to reformating the variables in a fashion similar to ITCN3, numbers each of the events and prints a header between each class.

Prints delta - Delta is a measure of the minimal distance from the cover found to the optimal cover possible. Were it not for trimming delta would be an exact measure of this distance.

Prints trim count - This is the number of times that intermediate stars had to be trimmed in the process of star generation. The sum of this and delta is an approximate measure of the maximum distance of the found cover from the optimal one.

Complex evaluation - This is a means of evaluating the goodness of the covers found by testing how well they cover a set of testing events. Various statistics are printed by the various evaluation routines to aid the user to determine if he should attempt to find a better cover with more training events.

Error handling - This is a subjective evaluation of how well the various versions handle mistakes in specifying events or control parameters. One of the things that weighed heavily was that if an

event was incorrectly specified (such as by leaving out a variable value or specifying an out of range value,) both AQ7 and ITCN3 would give no indication of this and attempt to process with invalid events which in my testing proved to be somewhat expensive. Incorrect specification of the control parameters also caused unpridictable results in these two versions. (i.e. if NGE were smaller than what was needed, selectors with no values were generated in the covers and the covering time was unduly long.)

TRIMMING CRITERIA

Since this is one of the major differences between the various versions of AQVAL/7 it will be handled seperately.

YAQ -A value is computed for each complex according to the following formula: VALUE = SUMFVW + TOTSW - SUMSW

SUMFVW = sum of the weights of the events covered by the complex.

TOTSW = total of all selector weights.

SUMSW = sum of the complements of the weights not used in the complex. The complement is relative to the largest weight of any selector.

This value is computed for each complex each time a trimming operation is done, be it for intermediate stars or the final star.

ITCN3 - The user may orde up to four criteria to be used one at a time as necessary each time either a star or an intermediate star needs to be trimmed. If only one criteria is necessary to

bring the (intermediate) star to cutstar complexes then only one is used. Otherwise those complexes left will be evaluated by the next criteria.

AQ5A - Intermediate stars are evaluated only according to number of events covered. If further itrimming is necessary it is done in an arbitrary fashion. The final star is evaluated according to the number of sellectors used in a complex if the event evaluation does not bring its size down to one event.

AQ7 - The user may order up to seven criteria to be used for picking the best complex from the final star. For intermediate star trimming only the first of these criteria is used. Further trimming is done in an arbitrary fashion if necessary.

TABLE OF FEATURE DIFFERENCES

FEATURE		YAQ	ITCM3	AQ5A	AQ7
Author	E YALOW	LARSON	TARASKI	MICHALSKI	
Language	360/ASM	PL/I	PL/I	PL/I	
Program size	10K	26K	18K	24K	
Ease of use	V GOOD	POOR	GOOD	FAIR	
Olist	yes	NO	NO	YES	
Criteria	yes	SOME	NO	YES	
LOST	yes	yes	yes	yes	
Negative events		YES	NO	NO	NO
Selector restrictions	NO	YES	NO	NO	

Formula updating	NO	YES	NO	NO
English output	NO	YES	NO	NO
Parameter echoing	YES	NO	NO	NO
Event echoing	YES	YES	YES	YES
Compactness of output	V GOOD	FAIR	POOR	POOR
Prints delta	YES	NO	YES	YES
Prints trim-count	YES	NO	YES	YES
Complex evaluation	GOOD	V GOOD	POOR	POOR
Error handling	GOOD	POOR	V GOOD	FAIR
No. of JCL cards needed		3	6	5	5
INFORM: VL	no	yes	no	yes	
INFORM: values	yes	yes	yes	yes	
INFORM: gamma		yes	no	no	yes

The principle advantages of each of the four aqval/7 programs are as follows:

YAQ -This version is the fastest and least expensive. The input specifications are the simplest. The output has the most compact format. Also the class covers produced are the simplest both in the number of complexes per cover and the number of selectors per complex. YAQ and AQ11 are the only versions to have an evaluation procedure. YAQ is the only version to support negative events. (a negative event is one whose class exclusion is known but who class membership is unknown.) YAQ is the only version to do any internal event reordering, a feature that contributes largely to its speed. YAQ has fairly good

error checking and is the only version to completely echo user input. in summary though, YAQ's principle advantage is its low cost and its speed.

AQ11 -Two thirds the cost of AQ5A and one third the cost of AQ7, AQ11 is the second most inexpensive version of aqval/7. It offers the widest variety of features, many of which are not found in other versions. The three most noteworthy of these are: the inputing of complexes to be updated by further training events, the specification of selector interdependencies, and the translation of output complexes into an english language equivalent. AQ11 also has the best cover evaluation routine.

AQ5A -The excellent error diagnostics are AQ5A's principle advantage. also the covers produced by AQ5A are slightly better than those of AQ7 and AQ11.

AQ7 -AQ7 allows a wide variety of user criteria specifications. It is also the only version that allows the user to specify an alternate order of class covering.

B. PERFORMANCE

II. TESTING PROCEDURE

I conducted eight comparison test of the above four versions of aqval/7 using six event spaces. Four of the event spaces consisted of approximately ten thousand events. Each of these had about 50 training events and the number of dimensions varried from 3 to 12. A fifth event space of approximately one million events had five

dimensions and 100 training events. The final event space, donated by jim larson, had about 10**25 events, 50 dimensions, and 286 events. Each event space had either three or four classes. intersecting covers were used throughout. Every event space was tested with a maxstar of one. Cutstar was set equal to maxstar in every test. The first event space and the last event space were also tested with maxstar=3. With a time limitation of 180 seconds neither 3O5A nor AO7 were able to complete the last test. The results of the tests are tabulated below:

KEY TO TERMS IN FOLLOWING TABLE.

PROG -PROGRAM NAME OF AQVAL/7 VERSION BEING TESTED.

CORE -THE AMOUNT OF BYTES OF MEMORY USED IN THE TEST.

TIME -THE AMOUNT OF CPU TIME CONSUMED MEASURED IN SECONDS.

SU -THE NUMBER OF SERVICE UNITS CHARGED BY CSO FOR THE TEST.

#CPX -THE TOTAL COUNT OF COMPLEXES IN ALL THE COVERS.

#SEL -THE SUM TOTAL OF SELECTORS IN ALL THE COMPLEXES IN ALL THE COVERS.

MAXSTAR -THE SETTING OF MAXSTAR USED. (CUTSTAR WAS ALWAYS SET EQUAL TO 0

MAXSTAR).

NV -THE NUMBER OF VARIABLES IN THE EVENT SPACE.

NCL -THE NUMBER OF CLASSES BEING TESTED.

NEVE -THE NUMBER OF EVENTS IN THE TRAINING SET.

VARSZ -THE SIZE OF EACH DIMENSION IN THE EVENT SPACE

SUMMARY OF TESTS

NV	NCL	NEVE	MAXSTAR	SPACE
7	3	52	1	10K
7	3	52	3	10K
3	4	54	1	10K
12	4	40	1	10K
5	4	46	1	10k
7	3	100	1	1M
50	3	286	1	10**25
50	3	286	3	10**25

SIZE OF EVENT SPACE IN ALL CASES IS APPROX 10K EVENTS

This first test was to show the relative performances for a medium number of variables of medium size with an average of 17 training events per class.

NV=7,NCL=3,NEVE=52,MAXSTAR=1,VARSZ=(4,4,4,4,4,5,2),SPACE=10K

PROG	CORE	TIME	COST(C)	#CPX	#SEL
YAO	47	.50	2	7	17
ITRN3	170	2.2	16	10	39
AQ5A	138	5.84	34	8	27
AQ7	142	11.54	64	13	28

The first test is repeated with a maxstar set equal to 3 in order that the tradeoffs between simplicity of covers and cost of finding them may be investigated. As can be seen the amount of improvement varried considerably at a fairly constant additional cost of 50 .

NV=7,NCL=3,NEVE=52,MAXSTAR=3,VARSZ=(4,4,4,4,4,5,2),SPACE=10K

PROG	CORE	TIME	COST(C)	#CPX	#SEL

YAQ	40	1.02	4	7	16
ITCN3	170	3.75	26	7	25
AQSA	138	8.59	48	6	15
AQ7	142	18.22	97	10	36

This test was designed to see what effect a small number of large variables would have. Since ITCN3 allows a maximum variable domain of 8 it could not be included in this test. The results show that YAQ's relative performance is at its worst here. Also its cost per cover is double that of the average for all other tests of this size event space. The reason for this drop in performance may be that the maximum cartesian distance between events is so small as to be negligible, thus causing the reordering of events to be more of a hindrence than an aid.

NV=3,NCL=4,NEVE=54,MAXSTAR=1,VARSZ=(16,16,10),SPACE=10K

PROG	CORE	TIME	COST(C)	#CPX	#SEL
YAQ	40	1.12	5	16	34
AQSA	138	7.14	24	11	27
AQ7	142	13.12	71	12	30

In a test of a large number of small variables and an average of 10 events per class YAQ's reordering is shown to maximum advantage. YAQ's covers were by far the simplest and its cost per cover was the cheepest of any of the tests.

NV=12,NCL=4,NEVE=40,MAXSTAR=1,VARSZ=(3,2,2,2,2,2,2,2,2,2,2,3),
SPACE=10K

PROG	CORE	TIME	COST(C)	#CPX	#SEL

YAQ	40	.52	2	8	21
ITCN3	164	2.69	26	11	48
AQ5A	138	5.94	34	11	39
AQ7	142	8.00	45	11	51

Here is another test of a medium number of variables of medium size.

NV=5,NCL=4,NEVE=46,MAXSTAR=1,VARSZ=(3,8,8,8,8),SPACE=10K

PROG	CORE	TIME	COST(C)	#CPY	#SEL
YAQ	40	.75	3	10	23
ITCN3	158	2.97	22	12	43
AQ5A	138	6.00	35	11	32
AQ7	142	14.97	81	14	30

AVERAGE OF THE ABOVE FIVE TESTS

NV=6.8,NCL=3.6,NEVE=48.8,MAXSTAR=1.4

YAQ	40	.78	3.2	9.6	22.2
ITCN3	165	2.90	22.5	10.0	38.7
AQ5A	138	6.70	35.0	9.4	32.2
AQ7	142	13.17	72.8	12.0	35.0

ABOVE NORMALIZED RELATIVE TO YAQ

ITCN3	4.12	4.12	7	1.04	1.74
AQ5A	3.45	8.59	11	.98	1.45
AQ7	3.55	16.88	23	1.25	1.65

Here the event space was increased 100 times and the number of training events doubled. Two things should be noted. 1) YAQ and AQ5A produced virtually identical covers. 2) The performance of AQ7 has degraded to 62 times the cost of YAQ.

NV=7,NCL=3,NEVE=100,MAXSTAR=1,VARSZ=(8,8,8,8,8,8,8),SPACE=10K

SPACE=1M

PROG	CORE	TIME	COST(C)	#CPX	#SEL
YAQ	40	.74	3	5	7
ITCN3	168	4.87	33	8	27
AQSA	124	6.17	36	5	7
AQ7	142	35.92	187	6	10

In the following four tests the average variable domain is 2.5 and the training set is 286 events. The event space is many times larger than any previously used. With a CPU time limit of 290 seconds only ITCN3 and YAQ were able to completely cover all three classes. The training set was donated by Jim Larson and was originally used to find a description of some rarte forms of cancer. As can be seen again YAQ produces a better cover for less cost. Also increasing maxstar produced only mildly better complexes at significantly more cost.

NV=50,NCL=3,NEVE=286,SPACE=10**25,MAXSTAR=1

PROG	CORE	TIME	COST(C)	#CPX	#SEL
YAQ	48	13.00	50	19	154
ITCN3	170	59.00	307	31	276

NV=50,NCL=3,NEVE=286,SPACE=10**25,MAXSTAR=3

YAQ	50	56.00	165	19	124
ITCN3	170	283.00	1428	25	248

NV=50,NCL=3,NEVE=286,SPACE=10**25,MAXSTAR=8

YAQ	62	113.00	464	18	93

NV=50,NCL=3,NEVE=286,SPACE=10**25,MAXSTAR=16

YAQ	84	610.00	2145	17	85

VI. CONTRIBUTION OF YAQ

YAQ is the fastest, smallest version of aqval/7 currently available. It also usually produces the best covers. Its speed and small size is principly attributed to its bein coded in assembly language. The finenes of the covers is the result of event reordering.

Assembly language allows for efficient subroutine management, efficient variable control, and error checking only in potential trouble spots.

Event reordering is YAQ's most important contribution to aqval/7. Reordering minimizes the number of intermediate stars produced in the course of star generation and also minimizes the number of selectors used in each complex generated. Cartesian distance is the unit of measurement used in event reordering. The cartesian distance between two events is the number of variables in which the two events have different values. Each time an event is chosen from F1 to generate a star around, the events from F0 are ordered relative to their cartesian distance from this event. Events from F0 are further grouped into those which are covered by the current intermediate star and those which are not. (the initial intermediate star is the entire event space.) then an extension of the chosen F1 event is made against the closest event from F0 still

covered by the current intermediate star. There are three major effects of this. One is that the extension is as small as possible since the two events differ in the minimal number of events. The second is that events from F^ cast a shadow. The closer the event from F^ the greater its shadow and the greater the likelihood that a maximal number of events from F^ will be excluded from the next intermediate star. (see diagram) this keeps the number of intermediate stars generated to a minimum. Since the resulting star is more generalized than that generated by the arbitrary processes used by the other versions of aqval/7, it is more likely that a greater number of events from F1 will be covered by the best complex which is what allows YAQ's covers to contain fewer complexes.

VII. AREAS FOR IMPROVEMENT OF YAQ

1. Add a weighted selector and event parameter so that particular events or variables will be covered or used first in star generation. This approximates some of the criteria used by AQ7.

2. Reorder the events in F1, the class being covered, so that the next star event is the one furthest from the last complex in the cover thus far generated. If the cover is null then use the event closest to any event in F^.

VIII. REFERENCES.

R>S>Michalski "Interval Generalization of Switching Theory"

Report No. 442

R. S. Michalski "On the Selection of Representative Samples From Large Relational Tables for Inductive Inference"

Larson Michalski "AQVAL/1 Users Guide and Program Description"

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-77-840	2.	3. Recipient's Accession No.
4. Title and Subtitle YAQ: A 360 Assembler Version of the Algorithm A^q and Comparison with other PL/I Programs			5. Report Date May 1977 6.
7. Author(s) Edward Yalow			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF MCS 74-03514
12. Sponsoring Organization Name and Address National Science Foundation Washington, DC			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

This paper contains a user's guide and program description of the program YAQ. Included are some examples of VL_1 formulas synthesized by the program and a comparison between the assembler version and PL/I versions of the covering algorithm AQ.

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

inductive inference
PL/I-BAL comparison
VL_1 formula
logical minimization

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

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