Cross-Classification Matrix Output Program

James H. Kuzirian, Kentucky Department of Transportation

ABSTRACT

State Highway Mileage data, giving the number of miles of existing roadway for all public roads, was classified by type of aid program, functional system, maintenance responsibility, and whether rural or urban. The resulting cross-classification required an output grid of 28 rows and 16 columns to present the data. However, the data is quite sparse, with typically 30-50% of the cells for a given county having any data. Therefore, a row and column index was computed for each data value, to locate it in the output page, using the "HPS" (full page) option under the FILE statement. Row and column headings were generated in an analogous fashion.

This system, including 6 SAS programs, was written in SAS/6.8D, it would have been easier and shorter to program in SAS71. The paper describes the report writing program in detail, which uses the following techniques: structured programming, link routines, HPS, FIRST, and LAST, variables, calculated variables for line numbers and print positions, "Interleave", SET statement within a loop, MERGER statement, and "IN-" parameter.

A recent production run consisting of 26487 input records took a combined total of 312.33 cpu seconds and 28.28 resident minutes on a 370/168, and resulted in 5390 values to be printed for 120 counties and a state-wide summary.

1. INTRODUCTION

1.1 THE PROBLEM

Picture, if you will, the following scenario, taking place at desk. After the usual preliminary chat, the client says:

I have a file of roadway sections for all public roads, the records of which contain the length of the section in miles, the average daily traffic on this section of road, and various codes with which it is identified. I wish to see a printout which shows, for each county, the number of miles of roadway for each intersection in the following matrix, (client then produces, with a flourish, a diagram which contains 16 columns across the top of the page, and 28 rows down the left-hand edge of the page.) After you get this, I want a summary for the entire state. Then, I want a repeat of the entire process for a second variable - the Annual Vehicle Miles Traveled (AVMT). The file contains 25,000 records, approximately.

The sample output pages (at the end of this paper - see figures 5 and 6) give some insight into the nature of the problem.

1.2 METHOD CHOSEN

The selection of a programming language quickly narrowed to a choice between FORTRAN or COBOL when the COBOL expert decided that the problem was not suited to COBOL. I selected SAS76 for the programming language because

1. HPS was the technique of choice for the output
2. Computations would require double precision to avoid roundoff error
3. Input could be done equally well in FORTRAN or SAS
4. Checkout facilities are adequate in SAS
5. Data set creation and access is easier in SAS

However,

1. FORTRAN (with the FLECS71 preprocessor) would have been superior for implementing the decision tables in the edit process
2. The FORTRAN subsetting facilities would have eliminated a few sorts

2. SYSTEM STRUCTURE

2.1 SYSTEM DIAGRAM

The system was broken into the components we see in figure 1:

- Input/Edit Program
- Computations
- Report Writers

3. INPUT/EDIT/COMPUTE

Since there is only one type of input record, the actual "INPUT" statement is quite straightforward.

Following the input statement there is a section of code which implements the decision table logic used in the program design. These decision tables edit the fields of interest in this application and compute the row and column index for the cell to which this record sums. See figure 2 for an example of this.

---

1 FLECS (FORTRAN Language Extended Control Structures) is a structured programming preprocessor for FORTRAN written by Terry Beyers, of the University of Oregon Computer Center, Eugene, Oregon.
Decision Table 11

\[
\begin{array}{c|c|c|c|c}
\text{colدخ} & \text{rurb} & 1 & 2 & 3 \\
\hline
\text{IF} & \text{A DEG} & < & 1 & \text{ELSE} \\
\text{OR} & 4 & < & \text{A DEG} & \text{ELSE} \\
\text{THEN} & \text{LINK CTRMAB1} & ; & ; & ; \\
\text{IF} & \text{A DEG} & = & 1 & \text{THEN} \text{colدخ} = \text{rurb} + 1 ; \\
\text{IF} & \text{A DEG} & = & 2 & \text{THEN} \text{colدخ} = \text{rurb} + 3 ; \\
\text{IF} & \text{A DEG} & = & 3 & \text{THEN} \text{table21} ; \\
\text{IF} & \text{A DEG} & = & 4 & \text{THEN} \text{colدخ} = \text{rurb} + 7 ; \\
\text{GO TO} & \text{TABLE21} & ; & ; & ;
\end{array}
\]

Figure 2: Decision Table Implementation Example

Finally, the Annual Vehicle Miles Traveled (AVMT) is computed for the road section represented by this record, and the record is output to a file called "SORTFILE", since these records are shorter than the original records.

2.3 COMPUTATIONS

An IBM SORT utility is used to sort the newly-created "SORTFILE" on county number, row index, and column index.

The next three programs use PROC MEANS and PROC SORT to accumulate totals. The first program uses PROC MEANS to compute road sums by county, row, and column; and writes these sums to the SAS file "CELLSUMS". The second program uses PROC SORT and PROC MEANS to compute row sums, column sums, grand totals, and various subtotals. This process results in seven data sets, which are concatenated into one data set, which is then sorted by county, row, and column. This concatenated sorted data set is then interleaved with the "CELLSUMS" data set to form the "ALLSUMS" data set. The third program uses PROC SORT and PROC MEANS to compute the statewide totals from the data in the "ALLSUMS" data set. The statewide totals are written into the "STATESUM" data set. At this point all calculations are completed.

2.4 REPORT WRITERS

Since the two reports to be produced are, for Existing Mileage and one for Annual Vehicle Miles Traveled, were not going to be used frequently, I felt that the most cost-effective approach would be to have two programs, one for each report, and pass over the final data set twice for report writing. This saved programming complexity and time, not only for this system, but also for a subsequent application against the same files.

The first of the two report-writers is the subject of this paper, and is discussed in detail below. The second report-writer (for AVMT) is in a copy of sections C, D, and F of the first, and needs no further elaboration.

One page is produced for every county which has any data. The data comes into the report writer sequence by county, row, and column. All heading information is printed once for each page.

\[\text{FIRST.CNTY.NUM} \text{ and \ text{LAST.CNTY.NUM} data values} \] are inserted into their proper place in the page image.

3. DOCUMENTATION METHODOLOGY

3.1 DOCUMENTATION PHILOSOPHY

At the root of my philosophy of documentation, and the motivating factor for the style of comments and commentary found in the program listing (available by contacting the author), are two observations which I glorified in 1974 as "Kuziian's Laws". Here they are in a slightly modified form:

1. (on documentation) "Documentation which has not been completed before the first program run will either never be completed or be worthless if it is."

2. (on program use) "Every program (or routine) which is used at least once will be modified and/or used again in some manner, regardless of any and all representations to the contrary by anyone."

3. (on clarity) "All code (and all documentation) more than 90 days old was written by someone else, even if my name is on it."

There are two points to consider for program documentation: What are the consequences of ignoring the implications of these laws, and what can we do, as programmers, to avoid the bad consequences? The format of the program listing for this report writer is a current expression of my attempts to improve the practice of program documentation by considering the program source language code and its associated commentary to be the primary source of essential documentation for running that program and modifying it, so that one does not end up with a source language deck without any clues as to who wrote it, what or who it was for, what it does, what data to use, in what format, what JCL, what masterfile, etc., provided at a minimum a cross-reference to the name and location of the best documented set of the contrary to anyone."

"One way to improve the quality of programs drastically is to take the view that the main purpose of a programming project is to write a highly readable manual describing a program. The program itself is merely a useful byproduct of this effort."

This is in harmony with my sentiments on the subject.

As a consequence of this philosophy, there will be many comments in my program listings, typically 50%—70% comment lines. The program discussed in this paper has an even higher percentage (82%), because a program structure diagram has been included in the comments (see figure 4, below).

I am aware that many people do not agree with this philosophy. My prayer is that I will never have to use, modify, or maintain their programs.

---

3.2 DOCUMENTATION TECHNIQUES

3.2.1 Identification

At the beginning of the program listing there is an identification and documentation section, followed by a "program structure diagram" (the latter is an optional item, included in this instance.) As items of identification, we begin with the module number, project label, system name, and program name, followed by who requested the program, who wrote it, month and year written, and the language used. Then there is a subsection describing the various input, output, and/or database files used by the program. The identification section concludes with a paragraph describing what the program does, and is closed off by a vertical delimiter. In this case two lines of " = * = ".

3.2.2 Documentation

The documentation section is introduced by the header "HOW TO RUN THIS PROGRAM," usually enclosed in some form of box. Detailed instructions are provided for execution of the program in either 150 or batch. These are heavily slanted toward the user, and include everything needed for execution of the program. If this seems to be too much detail, please remember that I consider myself to be a "dumb user" on this program after six months, and I am only trying to make it easier for myself.

3.2.3 Program Structure Diagram

This optional section is included in some form whenever the complexity of the program warrants it. Figure 4 pictures the data flow through a program which is included in the program listing. The advantages are that the diagram "pictures" the program design, disciplines the code development, and provides an index to the program which is very useful during program development and checkout.

3.2.4 Structured Programming

Structured Programming is primarily an attitude that one brings to the task of system and program design and coding. This can be most simply described by the "divide and conquer" philosophy, wherein functional blocks are decomposed into smaller units, until the implementation in computer language becomes manageable. Accomplishing this with some degree of success depends upon the programmer's general attitude toward the coding of the program design, discipline, and tolerance for complexity. With sufficient patience, the programmer can refrain from coding the uncodable (which is always covered over with more code) with a fairly low tolerance for complexity, the programmer will not allow a block of code to grow and grow and grow.

With the above principles in mind, one can do "structured programming" in SAS/DS or COBOL or even FORTRAN. A few SAS/DS statements facilitate this, most notably LINK, REUSE, but it is the attitude that one brings to the language, rather than the language itself, which is the controlling factor in most cases. I trust that the program listing does in fact display this. While the large program will only be seen on the structure diagram itself, the blocks and sub-blocks are cross-indexed within the body of code on comment cards.

3.2.5 Checkout Aids

Scattered throughout the program at various strategic locations (most notably at the end of a block of code), there are "checkout aids" which are intended to be activated by removing the "X" at the beginning of the line, and ordinarily to be used on a smaller "test" subset of data. When testing has been completed, the "X" is replaced, leaving the code available for emergency use by an uninitiated maintenance programmer.

4. PROGRAMMING TECHNIQUES USED

4.2 SET STATEMENT WITHIN A LOOP

The county names (to be printed on each page of the report) are read in from an external file of 120 observations (section 4.1). Since there are now 121 "counties" the name "STATEWIDE" is added as the 121st observation. This is done in section 4.2 by copying the county name table to a new dataset, one observation at a time, and upon encountering the end-of-file condition, to create one additional observation, and outputting it to the new dataset also.

4.3 MERGE STATEMENT AND "IN" PARAMETER

In section 4.1 the county name is appended to each observation in the dataset. If no data exists for a given county, an empty observation...
is created with only the county name and number, among the output section (D) to format and print a page with headers but no data. The "INP" parameter is used to signal that some data actually exists for this county; if no data values enter the observation from the "NEWSUMS" dataset, the observation is deleted.

This problem (of empty county pages) was discovered in a subsequent application of the program to a very small subset of the masterfile, for showing new road locations planned but not yet built.

4.4 DATA NULL

Program section D handles the page formatting and output, with headers but no data. The "DATA=Null" statement prevents creation of a SAS data set as we execute program statements for constructing the output pages.

4.5 "NPS"

The "NPS" option on the "FILE" statement holds all output on the current page until a "PUT PAGE=" statement is executed. This occurs when all of the data for the current county has been exhausted.

4.6 FIRST AND LAST VARIABLES

The "FIRST.CNTY_NUM" and "LAST.CNTY_NUM" variables indicate the beginning and ending of the data for each county. When FIRST.CNTY_NUM=1, the page headings for the new page are written by invoking section E with a "LINK" statement. When LAST.CNTY_NUM=1, the page is printed by executing the "PUT PAGE=" statement.

4.7 LINK, NESTED LINK

"LINK" statements are used in several places to invoke a body of code. For example, "FIRST-PAGE" is invoked once for each county; "PAGEBODY" is invoked once for each data value to be printed on the page. Several "LINK" statements are executed within each of these routines, some being executed within a loop. Some "LINK" routines are called from only one place; some are called from several places. See figure 3.

"LINK" is used when one has a functionally well-defined piece of code which (1) may be usable from more than one location, or (2) may be better maintained by clearly delimiting its boundaries by using this type of coding technique. Within this program, "LINK" routines are only nested to a depth of three.

4.8 LOOPS

Several short loops are implemented in program section E apart from the looping mechanism provided by the DATA statement. Those loops govern the placement of certain column and number headings in the row and column headings which are repetitive except for position. These loops are implemented with IF-THEN statements.

4.9 VARIABLES FOR LINK NUMBER AND PRINT POSITION

Figure 3 shows a section of code from program section F, for the placement of data within the page, together with the "LINK" routines which compute the row and column print position for each datum. The "LINK" routines were defined and first used in program section E for initializing the page headings, and therefore provide an example of such routines being used from more than one place in the program.

Since the correspondence between data row and link number, and column headings is not exactly regular, it was best to code the idiosyncrasies into a "black box" routine which can be called from anywhere in the program as needed.

4.10 LOGIC TESTS WITHIN SUBROUTINES

Logic tests in LINK routines COL_PP and ROW_LL (and many other places in the program) must form mutually exclusive and exhaustive subsets to operate correctly. Since IF-THEN-ELSE is not available in SAS76.

5. CONCLUSION

This application was a non-trivial example of the capabilities of SAS76 for personnel in our department, especially for report writing. Certainly, if SAS79 had been available at the time this system was written, its superiorities (over other programming languages) would have been evident in other portions of the system as well. More detailed information, including program listings, may be obtained by contacting the author at: Division of Technical Computing, State Office Building, Frankfort, Kentucky 40622.
**SECOND LEVEL**

A. INTERLEAVE-STATESUM-INTO-ALLSUMS
B. CREATE-COUNTY-NAME-TABLE
C. APPEND-COUNTY-NAME-TO-EACH-OBSERVATION
D. PROCESS-ALL-COUNTIES

**END**

A. INTERLEAVE-STATESUM-INTO-ALLSUMS
A.1 ASSIGN COUNTY NUMBER 121 TO STATEWIDE TOTALS AND INSERT INTO RECORD
A.2 INTERLEAVE DATASETS

**END-TO**

B. CREATE-COUNTY-NAME-TABLE
B.1 ACCESS COUNTY TABLE
B.2 ADD "STATEWIDE" TO COUNTY TABLE

**END-TO**

C. APPEND-COUNTY-NAME
C.1 MERGE CNTY_TAB WITH ALLSUMS BY CNTY_NUM

**END-TO**

D. PROCESS-ALL-COUNTIES
D.1 FOR EACH: CNTY_NUM
D.2 COUNTY-BREAK
E. INITIALIZIAGE-PAGE
F. WRITE-PAGE-BODY
D.3 COUNTY-BREAK

**END-TO**

E. OUTPUT-PAGE

**END**

F. WRITE-PAGE-BODY
F.1 COMPUTE-PRINT-POSITION
F.2 COMPUTE-PRINT-POSITION

**END-TO**

**END**

Figure 4: Program Structure Diagram

Figure 3: Portion of Section F Detail Code
### Figure 5: Typical County Report Page

#### Table 1

<table>
<thead>
<tr>
<th>State Maintained</th>
<th>Urban</th>
<th>Rural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toll</td>
<td>2,345</td>
<td>1,070</td>
</tr>
<tr>
<td>Other State Primary</td>
<td>3,209</td>
<td>1,040</td>
</tr>
<tr>
<td>State Secondary</td>
<td>5,170</td>
<td>1,490</td>
</tr>
<tr>
<td>Rural Secondary</td>
<td>37,040</td>
<td>5,900</td>
</tr>
<tr>
<td>State Private &amp; Service</td>
<td>20,330</td>
<td>1,230</td>
</tr>
<tr>
<td>Unclassified</td>
<td>1,590</td>
<td>4,140</td>
</tr>
<tr>
<td>Sub-total: State Maintained</td>
<td>32,345</td>
<td>6,070</td>
</tr>
<tr>
<td>Local Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>City</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Sub-total: Local Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Federal Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Private Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Grand Total</td>
<td>0.170</td>
<td>99,167</td>
</tr>
</tbody>
</table>

### Figure 6: Statewide Totals Page

#### Table 2

<table>
<thead>
<tr>
<th>State Maintained</th>
<th>Urban</th>
<th>Rural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toll</td>
<td>2,345</td>
<td>1,070</td>
</tr>
<tr>
<td>Other State Primary</td>
<td>3,209</td>
<td>1,040</td>
</tr>
<tr>
<td>State Secondary</td>
<td>5,170</td>
<td>1,490</td>
</tr>
<tr>
<td>Rural Secondary</td>
<td>37,040</td>
<td>5,900</td>
</tr>
<tr>
<td>State Private &amp; Service</td>
<td>20,330</td>
<td>1,230</td>
</tr>
<tr>
<td>Unclassified</td>
<td>1,590</td>
<td>4,140</td>
</tr>
<tr>
<td>Sub-total: State Maintained</td>
<td>32,345</td>
<td>6,070</td>
</tr>
<tr>
<td>Local Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>City</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Sub-total: Local Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Federal Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Private Roads</td>
<td>0.170</td>
<td>99,167</td>
</tr>
<tr>
<td>Grand Total</td>
<td>0.170</td>
<td>99,167</td>
</tr>
</tbody>
</table>