MANIPULATING LARGE HIGHWAY AND CRASH DATASETS

Elizabeth G. Hamilton and Carolyn D. Williams
University of North Carolina Highway Safety Research Center

ABSTRACT

Large data file manipulation using PC SAS® under DOS can appear to be impossible. However, using various techniques, many large file manipulations can be successful. Some of the techniques to be discussed include restricting size of variables with explicit length, limiting variables for a particular analysis with KEEP or DROP, pre-processing with PROC SUMMARY before PROC FREQ or PROC TABULATE, using STOP effectively while testing, using PROC DATASETS to delete intermediate work datasets, using BY option effectively, using PROC SORT only when necessary, and knowing when to use a tag sort.

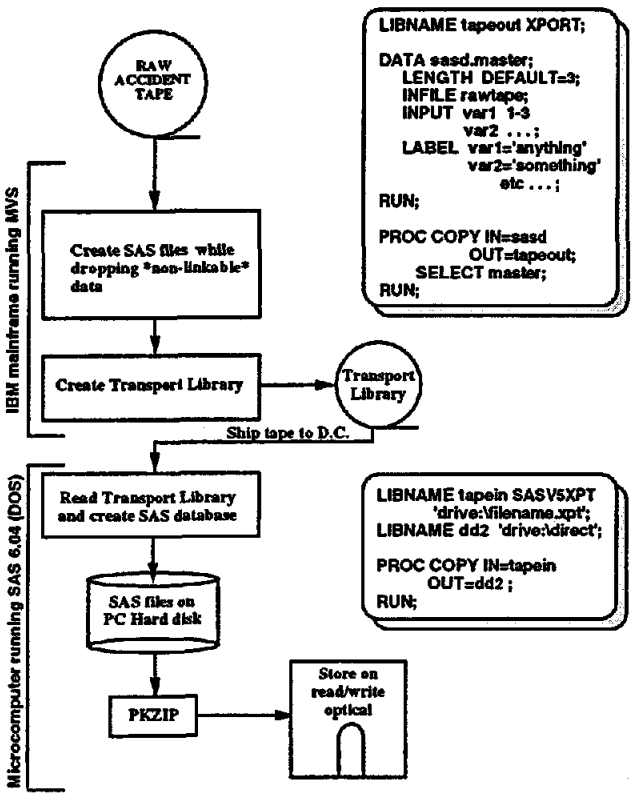
This paper will detail these techniques and their advantages and how the University of North Carolina Highway Safety Research Center uses them in processing accident, vehicle, roadway sections and curve data. These data files are part of a system called HSIS (Highway Safety Information System) which was developed by the Highway Safety Research Center under contract from the Federal Highway Administration (FHWA). HSIS is the primary database used by FHWA for its roadway-related safety analysis. It incorporates data from a select group of states - Illinois, Maine, Michigan, Minnesota, and Utah. The largest state file in our PC HSIS database has information for approximately 1.3 million vehicles, 755,000 accidents and 21,000 roadway miles. Matching this type of data to calculate crash rates requires a more complex merge than the usual MERGE BY. These techniques are transferable between the more powerful PC's (386,486) running DOS and IBM MVS operating systems (OS) running MVS/ESA and should apply to other environments.

INTRODUCTION

Regardless of the computer environment, conversion of large raw data files to SAS is a costly time-consuming endeavor. At the Highway Safety Research Center, we routinely perform raw data conversion or SAS data manipulation on datasets ranging in size from 500 thousand to 10 million observations. Usually, we use the Mainframe when processing files of this size. Recently, although we continue using the mainframe for the raw data conversion, we then transport the created SAS files to a PC for analysis and further processing. Using these large SAS files on the PC required that we develop methods to optimize data storage requirements and develop efficient SAS code. This presentation will detail some of the SAS features we employ as well as some of the techniques we developed and found useful.

We will limit our discussion to PC applications with some reference to the mainframe raw data conversion and the transfer of data to the PC. Benchmarks were run on an Everex 386/33 using PC DOS with SAS 6.04, on an IBM 486/33 using OS/2 SAS, and on an IBM 3090/170J mainframe using MVS SAS 6.07. Some of these features will be explained in detail along with examples of programs.

Typical HSIS File Creation



KNOW YOUR DATA

There are several types of data files which we routinely convert to SAS. The most frequently processed is "crash data". A raw automobile crash report will have detailed information on the accident (location, time, type of accident, weather, road condition, etc.), the vehicles involved (type of vehicle, make, model, year, damage, etc.), the people in the cars (injury severity, seating position, age, seatbelt usage, etc.). We store this information in three separate SAS datasets each having a different orientation. The accident-oriented crash file has one observation per crash. The vehicle-oriented vehicle file has one observation per vehicle and typically includes information on the driver. An occupant-oriented passenger file has one observation per person. Thus depending on the analysis issue, the needed analysis SAS files may need to be created in either Accident, Vehicle, or Occupant orientation.

Another type of data often used for particular studies in conjunction with crash data is a file that defines sections of roadway. A simple SAS merge is not sufficient for linking these files because an accident occurs at a specific point and the roadway describes a section of pavement of varying lengths. These segments can extend for several miles. Roadway datasets are an additional way of organizing the information, commonly referred to as segment orientation. Working with roadway data is called segment orientation. Other types of similar files include curve, grade, guardrail, intersection and interchange. Most of our analysis are done using one or a combination of these files.

Extracting a subset from a large data file is one of the most commonly performed manipulations on a dataset. Whatever platform being used, (UNIX, IBM, VAX, PC) or the type of data, the goal is to consider the limitations in your computing environment and minimize the impact on the items in shortest supply. We often are trying to trade off the impact of various techniques and their ability to reduce the size, the run time, the storage space, the CPU time, the understandability of the approach. CPU time translates into dollars and cents on the mainframe and 'TIME' on the PC. Whereas subsetting data may appear a relatively simple task, processing 430,000 observations and 53 variables on a PC requires careful selection of your subset criterion and thorough knowledge of the data you will be working with.

The first step in getting to know your data is a careful study of the reports from a few simple SAS procedures. The CONTENTS procedure is an easy way to get a list of all the variables in alphabetical order. To actually inspect a few records, we use the PRINT procedure and limit the number of observations. The MEANS procedure report gives information for the numeric variables (e.g. minimum value, maximum values, mean, number of missing). Knowledge of your data is imperative on any platform and size dataset. On a PC, you may want to 'walk that extra mile' to insure that when you design logic to process half million records, you have thoroughly attempted to locate any, if not all, data irregularities.

There are several subsetting methods available in SAS (IF, WHERE, and PROC SQL). Currently, only the IF and the WHERE statement are available in the DOS version. Using the IF statement, datasets can be subset using logical or ordered methods.

Example 1 - Logical subset using IF

```
DATA SMALLER;
   SET LARGE;
       IF (condition);
```

Example 2 - Ordered subset using IF

```
DATA SMALLER;
    SET LARGE;
      BY CASENUM;
          IF (condition);
```

Example 3 - Ordered subset using WHERE:

```
DATA SMALLER;
      SET LARGE;
        BY CASENUM;
           WHERE CASENUM LT nnn ;
```

Example 4 - Selecting records based on position

```
DATA SMALLER;
   SET LARGER (FIRSTOBS=nn OBS=mm);
```

Example 1 shows a simple IF condition. When converting raw data to SAS, this is the only subsetting method available. In examples 2 and 3, the subset selection is based on a logical condition of an ordered dataset. This method is used when you want your output dataset to remain in the original order. Example 4 creates a subset by extracting a block of observations from the middle. This is a quick way to get an extract for testing.

DATA STEP EFFICIENCY

Data step efficiency is not only a function of the size of the file you are processing, but also the size and complexity of the program to process the data. Regardless of the subsetting method chosen, the key to minimizing the frustrations of working with 'mainframe size' files on the PC is efficiency. There are several SAS tools which we have found very useful and have had positive affect on our ability to function well in the PC environment. We therefore offer the following suggestions.

1 - Benchmark program logic on a sample size of the dataset. Whenever possible use the aforementioned steps to create your sample size of data. This will make available to your program test data that encompasses a wide range of data values.

2 - Keep all variables in the sample data during design of program logic. Whereas one of the first rules of efficient programming is read and write only what you need, we have found that speed was not impaired by maintaining more variables in the sample subset. However we have discovered there is nothing more frustrating than testing a multi-step program and discover that some variables needed for a calculation are not in the dataset.

3 - Insert explicit RUN statements at the end of each data step so that the SAS LOG will be better organized. This log has very useful statistics which are helpful in detecting problems in logic, such as too many observations being dropped, data conversion, non-unique by variable merging, etc.

4 - Use multiple runs on the sample size, with PROC PRINT, PROC CONTENTS, PROC FREQ, and PROC MEANS. These are simple SAS tools but they can be very useful when developing complex SAS code.

5 - If your subset is going to be large (which is very likely because you could be processing a file with over 400,000 observations) consider some to the following examples.

The following code demonstrates methods to test your large files. Example 5 keeps almost all the variables in the subset and output messages to the log when calculated variables get out of the expected range. Later using PROC CONTENTS and PROC PRINT to view output datasets and determine what variables to keep for final production runs.

Example 5 - Keeping most variables while developing logic

```
DATA SUBACC;
   SET TESTACC(keep=varlist);

   IF ACCTYPE GE 4 AND ACCTYPE LE 12 ;
   TYPE= CLASS * 2;
   IF TYPE GE 56 AND LE 72 THEN
                        OUTPUT SUBACC;
   ELSE PUT 'CLASS VAR NOT IN RANGE ===>'
                        _ALL_;
RUN;

                  /* INSPECT VARIABLES */
PROC CONTENTS DATA=SUBACC;
   RUN;
PROC PRINT DATA=SUBACC (OBS=25);
   RUN;
```

Example 6 shows that multiple subsets (SUBACC and SUBERROR) can be produced during a single pass (or read) of the large file. It also demonstrates setting an explicit LENGTH for new categorical variables. Because these files are sorted by ACCYR, we can STOP the data step processing just after the last 1986 crash is read.

Example 6 - Create all subsets during one iteration.

```
      /* CREATE TWO SUBSETS            */
      /* 1- SUBACC - THE STUDY FILE    */
      /* 2- SUBERROR - ONLY THE DATA */
      /*      NEEDED TO CHECK ON       */
      /*      PROBLEM CASES            */

DATA SUBACC SUBERROR (KEEP=CASENUM
                  TYPE ACCTYPE CLASS);

   /* USE LENGTH STATEMENT TO LIMIT THE */
   /* AMOUNT OF STORAGE REQUIRED       */

      LENGTH TYPE 3. CODE $1;
      SET MIACC857 (KEEP=varlist);

   /* GIVEN THAT INPUT CONTAINS DATA    */
   /* FOR 1985-87, SORTED BY ACCYR:     */
   /* WE ONLY WANT 1985-86 DATA  IN STUDY*/

   IF ACCYR GT 86 THEN STOP;
```

```
/* CREATE 'CODE' AS A CHARACTER   */
/* VARIABLE TO SAVE DISK STORAGE  */
/* SPACE                          */

IF ACCTYPE GE 4 AND ACCTYPE LE 6 THEN
          CODE = '1';
     ELSE CODE = '2';
TYPE= CLASS * 2;

/* OUTPUT MULTIPLE FILES */

IF TYPE GE 56 AND LE 72
          THEN OUTPUT SUBACC;
          ELSE OUTPUT SUBERROR;
RUN;
```

Creating summary datasets is another method of handling large data files. Summarizing large data files that are used in repeated analysis is an excellent way to save disk space and time. We created summary datasets from a file containing 425,000 observations. The summary step on the 386/33 took 28.08 minutes, on the 486/33 running OS/2 only 5 minutes. We then ran frequencies on the summary files. The frequency on the summary took 4 seconds on the 386 and 0.96 seconds on the 486. Likewise, running frequency on the master file took 5 minutes on the 486 and 23 minutes on the 386. The biggest benefit of using summary datasets is realized not in the creation of the datasets but in time saved in repeated analysis runs using the variables in the summary dataset.

Data step efficiency is critical in PC SAS processing large files. When testing is complete, you only want to read what is needed, write what is required and only create numeric variables if they are to be used later for computation, otherwise use character variables. While SAS statements such as FREQ, LENGTH, DELETE, DROP, KEEP, and STOP OUTPUT are relatively simple they are very useful in coding efficient SAS programs.

USING SORT

There are several questions to ask yourself before sorting a file. The most important is, 'Is it necessary'? Sorts are very I/O intensive. It is not a procedure you would want to do on a large files unless absolutely necessary. A simple data step such as in Example 7 can determine if a dataset is sorted. Can the groups of data be processed with the NOTSORTED options? Is it possible to delay sorting until after the subset has been created? Unless your analysis requires by-group processing or descending, alphabetical listing, or some other type of multi-level processing, then a sort may not be required. It may also be that only the subset may need sorting. This should be determined earlier in the benchmarking process.

If you determine that sorting is necessary, the amount of time and resources required to sort a file depends on file size, number of variables and number of sort keys. We present three examples.

Example 7 - Is your data already sorted?

```
DATA SORTED;
   SET MIACC857(KEEP=SORTKEY);
      IF _N_= 1 THEN DO;
           KEY = SORTKEY; RETURN; END;
           RETAIN KEY;

      IF KEY GT SORTKEY THEN
           DO;
                PUT 'DATA NOT SORTED';
                STOP;
           END;
      KEY = SORTKEY; /* Prepare for next */
RUN;
```

Example 8 - Tag Sort

```
PROC SORT DATA=MIACC857 TAGSORT;
       BY YRCASE;
RUN;
```

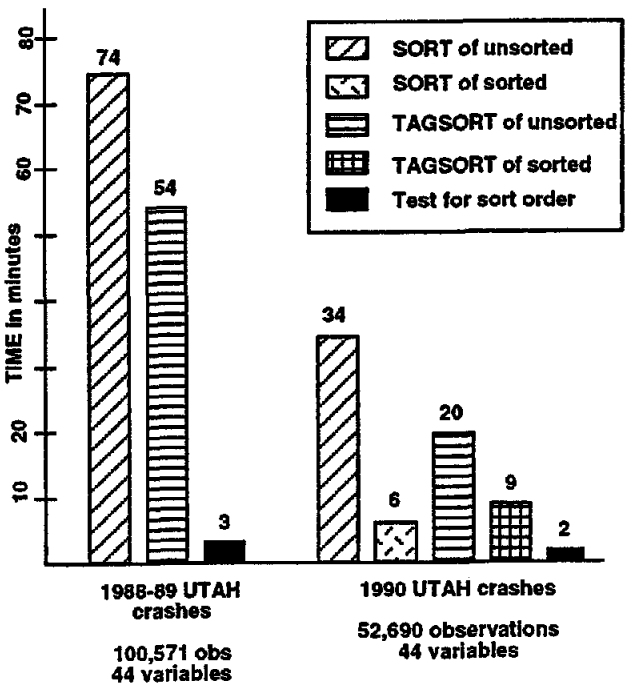
The TAGSORT option in PROC SORT is very useful when sorting large files with many variables because it does not shuffle all the SAS variables in the dataset along with the sort. It only uses the sortkeys in the sort process and reorders the data after the sort.

Example 9 - Limiting the variables

```
PROC SORT DATA=MIACC857  (KEEP=varlist)
                    OUT=MIACCSRT;
       BY YRCASE;
RUN;
```

A sort procedure with a keep statement specifying an output sorted dataset is the method we use most often. We have found that on large sorts the 'in-place sort' can run out of

Results from various SORT benchmarks



memory. Specifying the 'OUT=' option or further limiting the number of variables in the dataset should help with this problem. If not then example 10 uses the DATA step to create smaller datasets each having observations with the BY variable. These smaller datasets are sorted and combined. This will produce the same final results as a single large sort but take more steps and require more data management.

Example 10 - Divide and conquer as last resort

```
  /* SEPARATE ENORMOUS FILE BY YEARS */
DATA YEAR85 YEAR86 YEAR87;
     SET MIACC857;
          IF YEAR=85 THEN OUTPUT YEAR85;
     ELSE IF YEAR=86 THEN OUTPUT YEAR86;
     ELSE IF YEAR=87 THEN OUTPUT YEAR87;
     ELSE DELETE;
RUN;

    /* SORT EACH FILE SEPARATELY */
    /* USING YEAR IN THE SORT KEY */

PROC SORT DATA=YEAR85;
       BY YRCASE;
RUN;

    /* COMBINE THE SORTED FILES */
DATA MIACCSRT;
       SET YEAR85 YEAR86 YEAR87;
RUN;
```

COMBINING LARGE DATASETS

Combining datasets is a common technique used in the SAS system and is accomplished by using a SET, MERGE or UPDATE statement. The Set statement allows you to concatenate (stack datasets on top of one another). Missing values will be created for variables not common to both datasets). Using the SET statement with the BY options produces an interleaved dataset. Interleaving is very useful in large file processing because it allows the combining of files while maintaining the original order thus eliminating another sort. Merging files using the one-to-one or matched method assumes that the files being merged are in some sort order.

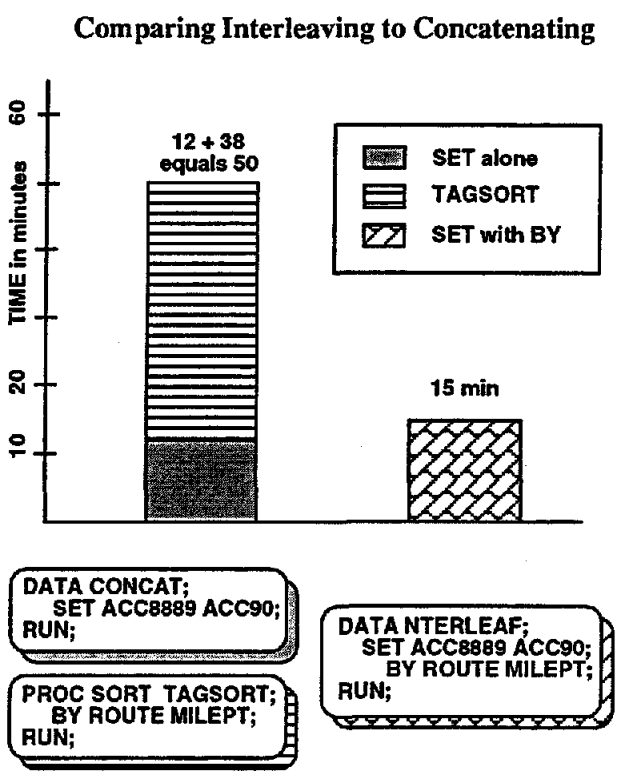
Merging large wide files are more difficult because you can easily run out of memory. The greater the number of variables in each input file the greater are the chances that memory will cause the program to ABEND. We suggest that if the end result is a dataset that does not require all of the variables from each file, then use the KEEP statement with each of the input datasets to save CPU and MEMORY. Also, run PROC CONTENTS on both input files. See if there are some numeric variables stored in 8-bytes that could have been stored in fewer bytes, or even in character format. If you have already gotten your final results, then it may not be timely to recreate the dataset. However, you could leave yourself a note with the program documentation to change the attributes if this job is ever re-run.

Example 11 - Concatenation

```
DATA COMBINED;
  SET ONE(KEEP=keepvar) TWO(KEEP=keepvar2);
RUN;
```

Example 12 - Interleaving *assumes ONE and TWO are already sorted

```
DATA COMBINED;
       SET ONE TWO;
            BY VEHNO SEATPOS;
RUN;
```



Example 13 - Merging /*wide datasets*/

```
DATA COMBINED;
      MERGE ONE (KEEP=keepvar)
            TWO (KEEP=keepvar2);

      BY VEHNO SEATPOS;
RUN;
```

DATA MANAGEMENT

Analysis files which may have taken weeks to develop are usually not just deleted when the job is done. Often these files are used randomly over a period of months or years. Managing, documenting and storing data is a critical issue regardless of the platform you are using. In the PC environment you have the added problem of needing the disk space on your disk to run other large jobs. Yet it is not feasible to reproduce a large analysis file each time new statistics are requested. We have found the following to be helpful in managing data on our PC:

1 - Organize your working and production files. For example, create directories by project.

2 - Use PROC DATASETS to modify or update directory information. This is great for eliminating work files when you are getting an out-of-space message.

3 - Whenever practical, store data in summary datasets.

4 - Create format libraries and store on diskettes, copying to hard drive only when needed.

5 - Store original master files offline on a large capacity storage device such as a read/write optical disk in compressed format.

6 - Be organized. Store analysis files in compressed format in project directories on optical disk.

7 - Delete unnecessary files from hard drive.

CONCLUSIONS

In conclusion, it is possible to process large files on a PC given some of the more sophisticated machines and technology. More attention must be given to efficient programming. In addition, the equipment should be some of the more powerful. For example, we would recommend at least a 386/33 or 486/33/50 processors, 16 bytes of RAM, 600 Mbytes of hard disk and a 650 Mbyte optical disk drive with three or more optical disks. This equipment does not guarantee quick turn-around time when processing large files but it will give you the capability of functioning 'mainframe-like' in a PC environment. Developing efficient SAS programming skills is the key to functioning effectively using any platform, on any sized dataset.

Author Contact:
Elizabeth G. Hamilton
UNC Highway Safety Research Center
134 1/2 E. Franklin St, CB#3430
Chapel Hill, NC 27599-3430
Internet: UHSREGH@unc.oit.unc.edu