

NEW DRUG APPLICATION TABLE WRITING STRATEGIES AND NOTES

Andy Barnett, Hong Li, Binita Modi, and Alfred Montalvo Jr.
Clinical Trials Support Services, Burlington, NC

Introduction

Presenting case report forms (CRF) for New Drug Applications (NDA) is often complicated by a number of problems, such as the methods for organizing the project and methods for the non-CRF-related tasks of collating, generating page numbers, and generating headers and footers (cf. Lue, 1992). These complications lead to several strategies, some of which we describe here. Two NDA formats are considered, which we will call "listings" and "tables" ¹. Both small to large numbers of CRFs and small to large numbers of observations will be considered. Additionally, the effects of deadlines are examined. Methods for generating headers and footers, generating page numbers, and collating output are explored, and it will be shown that these simpler tasks can be troublesome.

Definitions

For simplicity, we will assume that each CRF has been keyed as a corresponding SAS[®] dataset. Some parameters are defined as follows: page size (L), maximum number of SAS datasets that can be open at one time (S), maximum number of files that the operating system can have open at one time (F), maximum disk storage space (D), maximum RAM storage space (R), number of CRFs (C), number of patients (P), and average number of observations per CRF (K). In conjunction with these parameters, the term "order" is defined as a measure of resources required ²; linear order is considered fast, polynomial is tractable, and exponential is intractable. The current work is based on SAS 6.04 for PCs using MS-DOS[®] 5.0.

Listings are defined as grouped BY CRF INVEST DRUG PATIENT VISIT (p. 81, FDA, 1988) ³; a new page occurs on FIRST.CRF. On the other hand, tables are grouped BY PATIENT CRF VISIT (see Example 1)⁴; new pages occur on FIRST.PATIENT, and often on FIRST.CRF due to available space on the page. Tables inherently require more work than listings because the largest table by-group variable, PATIENT, has data in all CRF files; thus, data from all CRF files

must be interleaved to produce tables. If C is small, this extra work is negligible; but if $C > S$, then several steps are needed just to interleave the data. Also, tables often produce more pages (CK/L) than listings (PC). In general, we will consider C and K to be large, since then the difficulties are not trivial.

Example 1: Table with two CRFs

PATIENT: 1638/SLP	DRUG1	SEX: FEMALE
RACE: WHITE	12/03/51	AGE: 40

ADVERSE EXPERIENCE

VISIT	VISIT DATE	DATE OF ONSET	RELATED (*)	SIDE EFFECT
1	07/01/92	06/25/92	1	DRY MOUTH
2	07/08/92	06/30/92	3	MUSCLE STIFFNESS
		07/02/92	2	DIARRHEA

ADVERSE EXPERIENCE - DRUG RELATIONSHIP COMMENT

VISIT	VISIT DATE	SIDE EFFECT	COMMENT
2	07/08/92	MUSCLE STIFFNESS	DRUG REDUCED

* Relationship to drug:
1 = Definite 2 = Probable 3 = Possible
4 = Unlikely 5 = None . = Missing

PATIENT 1638 PAGE-1 APPENDIX B-24

Problems of Production in General

Both the production process and final product should have several characteristics to make them usable. Conway and Gries (1977) list five performance measures: speed, space, readability, subsequent ability to be modified, and production time. These can be reduced to simplicity, performance, and flexibility. The most important is simplicity. If consumers cannot understand the work, then one cannot expect consumers to be able to trust or modify the work, making it useless. However, the program must perform and be flexible if it is to be completed, so simplicity sometimes must be compromised to some extent. These three

criteria will be used to examine the strategies that follow.

Specific Problems of Production Projects

Two specific problems are expected to occur during production. First, deadlines often will be tight, so dividing the work will help. However, if the work is divided, it must be recombined at some point, creating a communication constraint. Second, mistakes will be discovered throughout the process, and those errors that are discovered late need to be resolved quickly. Thus, the ability to produce "partial tabulations" (i.e., a few rather than all CRFs) is desirable.

Specific Table Writing Problems

Table writing must solve three specific problems, namely collating⁵, page numbering, and generating headers and footers. These are not so much related to the CRFs as they are to more general tables and reports, but they are perhaps the most troublesome part of table writing. Collating can take considerable computer resources. Page numbering would be less of a problem if errors were not discovered late in a project; when an error is discovered, the affected portion must be rewritten, possibly affecting all subsequent page numbers, and thus perhaps requiring much reprinting. Additionally, if each version of a project is retained, the output can use up space quickly⁶. Finally, the methods available to generate headers and footers each have certain advantages and disadvantages, as described below.

Solutions to Specific Problems

Two schemes can be used to collate files, called the "linear" and the "tree" schemes after the manner in which datasets are collected⁷. Both schemes recognize a limit to the number of files, S , that the system can open at one time; if the number to be collated (in our case this is the number of CRFs, C) exceeds S , then several SAS steps are required to complete the task. Before considering these schemes, three useful terms are defined. A "group" is a set of files that are all collated in a given SAS DATA step. A "phase" or "pass" is a set of groups (SAS DATA steps) that uses all the output from the previous phase (or the original datasets if the phase is the first phase).

The linear scheme groups datasets in a linear fashion, where the output from each step is one of the inputs to the next step; it can be pictured as follows:

$$(\dots(((K + K) + K) + K) + \dots + K).$$

Note that there is only one phase (one pass). The time required (complexity) is of order $O_T = K C^2$, and the space complexity is of order $O_S = 3KC$.

A better strategy is the "tree scheme", where each phase groups all output from the previous phase in groups of the same size, until only one file is output. Each phase corresponds to a level (also called a depth or generation) in the tree, and each phase contains several steps (except the last one which has one step). No phase has a dataset repeated. This method is pictured as

$$(\dots(\{ \{ (K+K) + (K+K) \} + \{ (K+K) + (K+K) \} \} + \{ \{ \dots \} + \{ \dots \} \}) + \dots).$$

Here, the time complexity is $KC \log_G C$, where G is the maximum number of datasets per group. The space remains $3KC$.

In both the linear and tree groupings, increasing the size of the groups reduces time by reducing the number of steps. Also, in both methods, when $C \leq G \leq S$, time becomes linear (KC), and space drops to $2KC$ (a 50% drop) because the interphase copies do not exist (there are only one phase and one group, and thus one DATA step)⁸.

Two implementations can be used to collate files, depending on whether the files are text files or SAS datasets⁹. If the files are text files¹⁰, then a number of INFILE statements (up to some limit, S , which depends on such things as R and F) can be used in one step to collate the files. Multiple steps are needed if $C > S$. The I/O time required is of order roughly $K C \log_S C$, and the disk space required is of order $3KC$. If $C \leq S$, then the time drops to KC and space to $2KC$ (since $\log_S S = 1$, and there is no "in-between" interleave step¹¹.) If the files are SAS datasets, then a similar number of files, S , can be interleaved in one or more SET statements in the same step (p. 251, SAS Institute Inc., 1988). The I/O time and space complexities are the same as for INFILE. With prior PROC SORTs, these multi-step interleaves are similar to a multi-phase¹² sort/merge in the sense of Miller (1987). When a multi-step interleave is used, one should group files as many

as possible, and never use the same dataset twice in the same phase. This grouping produces the logarithmic component in time complexity ¹³.

Like collating, page numbering also can be arduous. Two basic methods for generating page numbers can be employed, depending on whether the data are merged as text files or as SAS datasets. If text files are generated first, then a post-processor using INFILE can collate (an INFILE-interleave) the text files and generate page numbers on the last phase. If SAS datasets are being interleaved prior to the text generation, then the page numbers would be generated in the same program that generates the text. A variant would generate text-like SAS datasets for each CRF table, and these would be interleaved using a multi-phase SET-interleave, with the last phase generating the table text file instead of SAS datasets.

In either case, it is not necessary for programmers to share information about their page numbers, since the page numbers can be generated last. A more compelling concern is that once the final table text file is printed, there is a fair chance that an error will be discovered, and the correction may affect all subsequent pages simply because the page numbers have changed. Some insurance against this problem can be obtained by requiring that the maximum number of pages used for any patient for a particular CRF be used for all other patients for that CRF. This fixed paging significantly reduces the chance that a complete reprint will be required if a corrected CRF for some patient requires more or less pages; a complete tabulation will be required only if the new maximum number of pages exceeds the previous maximum for that CRF.

Finally, the generation of headers and footers can present difficulties if one is not careful. Headers are easy to generate automatically using FILE PRINT HEADER= to specify a header routine to execute whenever a page break occurs ¹⁴. However, footers cannot be printed so easily. One way to generate footers is to print the footer in the header routine, using PUT # to print on the correct lines (p. 315, Aster and Seidman, 1991). This requires a PUT # to be done after the footer is printed, so that the detail lines will be in the right place. But this creates the danger of overrunning the footer with detail lines, since the detail lines are printed after the footer. Thus, a better way of printing footers is as follows: before printing a detail by-group (e.g., a VISIT), make certain there is enough room on the page to print the detail line(s), and if not

then print the footer and PUT _PAGE_. (To determine how many lines are needed to print all records for each visit, merge the N-statistic output of a prior "PROC MEANS NOPRINT; BY PATIENT CRF VISIT; OUTPUT OUT=KVISIT N=KVISIT;" with the CRF file [Barlow and Ringelberg, 1992].) Another method for generating headers and footers is to keep track of the remaining lines on the page and LINK to a HEADER or FOOTER routine at the appropriate times using the LINESLEFT variable, LL, with the FILE PRINT N=L LINESLEFT=LL statement (p.303, Aster and Seidman, 1991) ¹⁵. We have found that it is best to print the header, detail, and footer in that order.

Some General Strategies

Several strategies for solving the production problem exist, three of which are given here. The first, called the "SINGLE" strategy, uses one program to tabulate all CRFs. The second, called the "MULTIPLE" strategy, has multiple, self-contained programs, each one tabulating a single CRF ¹⁶. The third, called the "HYBRID" approach, is a derivative of the first two; it has one main program that INCLUDEs separate code segments, one for each CRF; the main program performs the collation and header/footer generation.

The SINGLE strategy has various advantages and disadvantages. Although it is very simple, it tends to get massive for large C. Dividing the work among several programmers is possible, but it is a rather difficult task to combine all the code segments and make them work together. This results in a need for substantial communication among the programmers, and the establishment of conventions that may interfere with programmer style. The programmers are not effectively isolated. This need for communication increases the chance that problems in recombining their code will be discovered too late to meet a deadline. The problems of collation, page number generation, and partial tabulation are reasonably easy to solve using the methods described earlier, as they are with the other methods. Overall, the SINGLE strategy is not a particularly powerful approach, and is best used when C is small.

The MULTIPLE approach is more powerful. Although there are more files, the problem is conceptually simpler, since the multiple programs logically split the problem into pieces. The work is easily divided among programmers; each programmer simply solves a smaller problem of tabulating a particular CRF. The need for

communication between programmers is considerably reduced, so they have greater freedom in choosing methods. However, their results must be recombined at some point, and well-defined guidelines for input and output must be given each programmer if their results are to be integrated properly. Partial tabulations are simple, since a single program is involved, and no change to other CRF programs will be needed. The MULTIPLE approach has a peculiar twist, in that it must generate text files since each CRF program operates independently and the text is CRF-dependent. The problem that develops is that one has no choice but to interleave text files, rather than SAS datasets as in the other methods¹⁷. Furthermore, the fact that text files are created prior to collating requires that only one CRF be printed per page (a minimum of PC pages)¹⁸. In either case, page numbers are easy to generate.

The HYBRID approach is perhaps the most complicated, but has a number of advantages. The INCLUDE files make dividing the work an easy task. The INCLUDE files also facilitate communication and establish conventions that can be easily modified and redistributed. The communication is further enhanced by the fact that a main program, which no one modifies, is shared by all the programmers. The INCLUDE structure and main program act as an interface between the programmers before their code is assembled for the final run. This early, in-depth, automatic communication provides insurance that, when the code is assembled, little if any change will be required. Thus, the programmers are isolated effectively and have considerable freedom in the methods they choose. The problems of partial tabulation, collation, and page number generation are solved easily using any of the methods described earlier.

The SINGLE and HYBRID approaches share two problems appearing when files are collated, name collision and type avoidance. When an interleave is performed, a name collision occurs if two different CRFs share variables of the same name. In the worst case, the program could abort if the two variables are of different types. This suggests a naming scheme is needed to prevent collision. At the same time, enormous space can be wasted by variables whose names appear in one CRF and not in any others, although they may be of the same type and length as variables in other CRFs. This problem can also be solved with a renaming scheme. The method we propose is to assign each CRF a number, and to rename

each numeric variable as N1, N2, ..., N_n, where *n* is the number of numeric variables present in that CRF. If the numeric variables have to be of different lengths, then append a number designating the length before the index (e.g., N8₁). Similarly, rename character variables as C_l_i, where *l* is length in bytes and *i* is *i*th distinct character variable of length *l*. This will minimize the number of variable names and allow variables with similar attributes to be merged. The obvious disadvantage of this method is that the name says nothing about the meaning of the variable; perhaps an auxiliary variable could be kept for each of these containing the name of the original variable; this would be constant within each CRF by-group.

Another problem shared by the SINGLE and HYBRID approaches is the possible need for additional BY-variables in some CRFs (e.g., repeat labs at the same VISIT). This can be solved as shown below:

Example 2: A method to generate a FIRST variable

```
(A) SET TABLES;
    BY PATIENT CRF VISIT N;

(B) SET TABLES;
    BY PATIENT CRF VISIT;
    LN=LAG(N);
    IF FIRST.VISIT OR N NE LN THEN FN=NOT 0;
    ELSE FN=0;
```

Since the BY-variables are limited to BY PATIENT CRF VISIT, each programmer might generate variables LN and FN as shown above in Example 2B. Then, FIRST.N from Example 2A is the same as FN in Example 2B.

One advantage shared by the SINGLE and HYBRID methods is that header/footer generation occurs in one place, and thus in a consistent manner. The MULTIPLE method could use its post-processor to keep them consistent; but as we have described it, only page numbering and collation are performed last, not header/footer generation.

Comparing the three methods, we find that the HYBRID method offers the most choices and contains the most insurance. Although the HYBRID method is more complex, dividing work is easy, and the resultant costs of communication are reduced by using INCLUDEs and a main program to aid communication and establish and distribute conventions. Programmers are much freer than they are in the SINGLE approach, although not as free as with a MULTIPLE approach.

Partial tabulations with HYBRID are as straightforward as with MULTIPLE and much easier than SINGLE. Like SINGLE, HYBRID can use any method described above for collation and page number generation. On the other hand, the MULTIPLE approach tends to require that an INFILE-interleave be used, and that CRFs appear on separate pages, both because text files are generated before collation occurs; also, page numbers must be generated using text as input.

A few problems might be expected in the worst case with these approaches. If communication among programmers breaks down too much, then their work cannot be integrated without extensive modification and time (this is less of a problem with the SINGLE and HYBRID approaches, since they have conventions built-in). When $C > S$, then if the size of the output, roughly KC , exceeds $D/3$ (or $D/2$ when $C = S$), then the interleave will not work. More complicated algorithms are needed to interleave in this case.

Conclusion

In summary, the most difficult tabulation tasks have little to do with CRFs. Collating, generating headers and footers, and page numbering can cause serious complications. When computer time and space are strained, a tree interleave is preferred, using as many files per group as possible. It is unclear which of INFILE or SET is more efficient. Splitting the work among several programmers to meet deadlines, and letting them write their own code for each CRF cuts down on the time and communication needed to finish the project, although adequate input/output guidelines must be given to each programmer to allow their work to be integrated. The HYBRID approach appears to have the best features and greatest range of choices. With these considerations in mind, a large project must be designed carefully if it is to be on-time and usable.

Footnotes

1. Listings are similar to tables and involve less work, so they are not covered in detail. See Figure 4, p. 767, Barlow and Ringelberg (1992), for a sample listing, and Fig. 3 for listing program.
2. For a definition of "order", see p. 292, Conway and Gries (1975).
3. The FDA says "the usual presentation is by patient and, for each patient, by visit, with patients grouped by

study, by investigator, and by treatment." This is the same as BY CRF INVEST DRUG PATIENT VISIT, or what we have defined as listings. Listings do not have the problems of collating because the CRFs fit together sequentially, and so listings are less difficult than tables.

4. Note that each patient undergoes one treatment, so BY PATIENT DRUG is the same as BY PATIENT.

5. When preceded by sorting, the operations of "collating" as used in everyday parlance, and "interleaving" as described on pp. 251-252, SAS Institute Inc. (1988), are synonymous with the "sort/merge" operation as described in Miller (1987).

6. One way to cope with versions of output is to keep only the old programs. But this is acceptable only if the old output can be regenerated quickly enough without tying up resources.

7. These schemes correspond to graphs. See Even (1979) for a discussion of graphs.

8. Derivation of complexities is as follows: For the linear method, note that each group has the previous group (or the first dataset in the case of the first group) plus $G-1$ more datasets, and there are $N = \text{CEIL}((C-1)/(G-1))$ groups. Thus, the I th group has $(G-1)(I) + 1$ original files (unless it is the last group, which has C original files). The total I/O is then:

$$K(C + \sum_{I=1}^{N-1} ((G-1)(I) + 1)) = K((G-1)(N^2 - N)/2 + N - 1 + C)$$

Thus, O_T is order $K C^2$ for all G and C ; for $G=2$, $O_T = K(C^2 + C - 2)/2$. O_S is measured by noting that the last group has $K(C-1)$ records from the previous group plus K records from the last file, all of which must be output, plus the original KC records, making a total of $3KC$ records on disk. For the tree interleave, each phase divides the output from the previous phase into groups of G files; this division produces $\text{CEIL}(\log_G C)$ phases, and each phase passes over all the data exactly once (KC records), for a total of $KC(\text{CEIL}(\log_G C))$ operations; this is of order $C \log C$. The space required is the same as for the linear grouping, in that each intermediate phase needs space for the KC original records, for the output from the last phase which it uses as input, and for its output, for a total of $3KC$ records (see p. 183, Miller, 1987).

9. The sort time and space complexity is not examined here since it cannot be varied using PROC SORT.

10. A text file differs from a SAS dataset in that a text file has record structure only, with every record containing only character data. A SAS dataset breaks records further into variables, each of which may be one of a variety of forms. Thus, text files can be printed without further processing, whereas SAS datasets must be processed to make sense when output.

11. Note that INFILE apparently closes any other infile and opens the current infile, causing the infiles to be opened and closed roughly CK/L times.

12. The terms "step" and "phase" are not interchangeable. A step is a SAS DATA step and corresponds to a group, whereas a phase consists of several groups in one level.

13. Due to space limitations, we have omitted the code for these multi-step INFILE- and SET-interleaves. Interested users can contact us for copies of code that will perform a multi-step interleave using either INFILE or SET.

14. Aster and Seidman (1991), p. 192, suggest that using HEADER= is often not advantageous.

15. If a FOOTER routine is used, we suggest that one use "FOOTER;; DO WHILE LL>1; PUT; END; PUT footer; RETURN;" rather than PUT #pagesize footer;", because anytime the PUT # line control is used, there is a chance of overwriting previously written data (in this case one might accidentally overwrite detail lines that were written in the wrong place, which might easily go unnoticed).

16. We have not attempted the MULTIPLE approach.

17. Whether an INFILE interleave takes more resources than a SET interleave is unclear. However, if an INFILE interleave does require too much resources, then it is conceivable that the CRF programs could write text-like SAS datasets (TLSDs), containing perhaps a single, long, character variable, where each observation is a line of text; these TLSDs could then be interleaved with a SET statement.

18. This requirement could perhaps be relaxed by adding "token" lines that tell a post-processor what PATIENT and CRF the data that follow are from, and how many lines must be printed on the same page (generally this is roughly the number of records in a VISIT by-group). We have not attempted this approach, however.

Bibliography

Aster, Rick, and Seidman, Rhena. *Professional SAS® Programming Secrets*. USA: Windcrest, 1991.

Barlow, Ginger, and Ringelberg, Angela S. "Dances with Data." *Proceedings of the Seventeenth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc., 1992.

Conway, Richard, and Gries, David. *An Introduction to Programming*. Cambridge, MA: Winthrop, 1975.

Even, Shimon. *Graph Algorithms*. Rockville, MD: Computer Science Press, 1979.

Food and Drug Administration. *Guideline for the Format and Content of the Clinical and Statistical Sections of an Application*. U.S. Department of Health and Human Services, July, 1988.

Lue, John C. "Generating Statistical Tables Meeting NDA Guidelines the Lazy Way." *Proceedings of the Seventeenth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc., 1992.

Maurer, Hermann A. *Data Structures and Programming Techniques*. Trans. Price, Camille C. Englewood Cliffs, NJ: Prentice-Hall, 1977.

Miller, Nancy. *File Structures Using Pascal*. Menlo Park, CA: Benjamin/Cummings, 1987.

SAS Institute Inc. *SAS® Language Guide for Personal Computers, Release 6.03 Edition*. Cary, NC: SAS Institute Inc., 1988.

Acknowledgments

SAS is a registered trademark of SAS Institute Inc.

MS-DOS is a registered trademark of Microsoft Corporation.

Address correspondence to:

Andy Barnett
Clinical Trials Support Services
P.O. Box 1815
Burlington, NC 27216-1815
(919)222-8030