Chapter 2

Fellegi-Holt Systems

Chair: John Kovar, Statistics Canada

William E. Winkler ◆ Thomas F. Petkunas

Joel Bissonnette

William E. Winkler ◆ Lisa R. Draper

Chapter 2

DISCRETE: A Fellegi-Holt Edit System for Demographic Data

William E. Winkler and Thomas F. Petkunas,
U. S. Bureau of the Census

Abstract

This document provides background on the workings and an application of the DISCRETE edit system. The system is a prototype whose purpose is to demonstrate the viability of new Operations Research (OR) algorithms for edit generation and error localization. While the OR algorithms are written in a general fashion that could be used in a variety of systems, the i/o, data structure, and imputation sections of the code are written in a survey-specific fashion. The source code cannot easily be ported to a variety of computer systems and is not easy to maintain. The first two sections consist of a description of the basic edit system and an example showing specific details of the input and output files used by the software. The final section is a summary.

DISCRETE: A Fellegi-Holt Edit System for Demographic Data

William E. Winkler and Thomas F. Petkunas,
U. S. Bureau of the Census

Description of the DISCRETE Edit System

The following subsections describe aspects of the DISCRETE edit system.

Purpose, Model, and History

The DISCRETE edit system is designed for general edits of discrete data. The system utilizes the Fellegi-Holt (FH) model of editing. Source code for DISCRETE was written by the author (Winkler, 1995a) and is based on theory and computational algorithms from Fellegi and Holt (1976) and Winkler (1995b).

Software and Computer Systems

The software consists of two programs, gened.for and edit.for. The software is written in FORTRAN and is not easily portable. With some work, the software runs on IBM-PCs under DOS and UNIX workstations. The programs run in batch mode and the interface is character-based.

The first program, gened.for, generates the class of implicit edits that are necessary for the error localization problem. The error localization problem consists of determining the minimum number of fields to impute so that an edit-failing record will satisfy all edits. It uses as input a file of explicit edits that have been defined by an analyst. As output, it produces the file of implicit edits that are logically derived from the explicit edits and also checks the logical consistency of the entire set of edits. The class of implicit edits that are generated are so-called maximal implicit edits. The class of originally defined explicit edits plus the class of maximal implicit edits is known to be sufficient for solving the error localization problem (Garfinkel, Kunnathur, and Liepins, 1986, hereafter, GKL) and known to be a subset of the class originally defined by Fellegi and Holt. The method of generating the maximal implicit edits is due to Winkler (1995b) and replaces an earlier method of GKL. The GKL edit-generation algorithm has a driver algorithm for traversing nodes in a tree and an algorithm for generating new implicit edits at each node in the tree. The nodes are the locations at which new implicit edits can be generated. The Winkler algorithm has a different driver algorithm for traversing the nodes in the trees, an in-between algorithm that determines the subset of edits that are sent to the implicit-edit-generation algorithm, and an edit-generation algorithm similar to the one of GKL.

The second program, edit.for, performs error localization (i.e., determines the minimal number of fields to impute for a record failing edits) and then does imputation. The input files consist of the set of implicit edits produced by gened.for and the data file being edited. The error localization algorithm

(Winkler, 1995b) is significantly faster than an error localization due to GKL because it first uses a greedy algorithm and then, if necessary, uses a cutting plane algorithm. Error-localization by GKL is via a pure cutting plane argument which is orders of magnitude slower than the greedy algorithm even with moderate size problems. While greedy algorithms can yield suboptimal solutions with general problems, greedy algorithms typically yield optimal solutions with edit problems. Cutting-plane arguments are generally known to be the most effective for solving integer programming problems (Nemhauser and Wolsey, 1988). Another difference between Winkler (1995a) and GKL is that the number of edits passed to the error localization stage grows at a somewhat slower exponential rate under Winkler (1995b) than under GKL. The slower exponential growth is due to a more precise characterization of the implicit edits needed for error localization (Winkler, 1995b). As computation in integer programming is known to grow faster than the product of the exponential of the number of edits and the exponential of the number of variables associated with the edits, the new error localization procedure should be much faster in practice. The imputation module of edit.for currently delineates the set of values for the minimal set of variables needing to be changed so that all edits are satisfied. In applications of the DISCRETE edit system, the imputation methodology currently consists of analyst-defined if-then-else rules of substitution. The substitutions for edit-failing data satisfy the edit rules and are very survey specific. Although general substitution rules within the restraints imposed by the Fellegi-Holt theory could be developed, they often would not be as acceptable to subject-matter specialists as the survey-specific rules. The advantage of the general substitution rules is that they would greatly speed the implementation on new surveys because analysts would not have to spend as much time defining edit rules and substitution rules.

The outputs from the second program consist of summary statistics, the file of edited (i.e., containing imputes) data, and a file giving details of each record that was changed. The details consist of the failed edits, the minimum fields to impute, and other information related to specific data records.

Documentation

The only documentation associated with the DISCRETE edit system is Winkler (1995a). The documentation is minimal and only describes how to compile and run the software on the example included with it.

Limitations

As computation grows exponentially as the number of variables and the number of value-states of variables increase, large systems of edits may be slow. At present, we do not know the the larges size the system will handle. The system, which has i/o modules based on an earlier system that utilized algorithms of GKL, does not easily recompile and run. A large number of include files must be modified and initial values of some data structures that describe the data are hard-coded.

As the software is an early prototype version, insufficient time has been spent on debugging source code. While the OR portions of the source code run perfectly on a variety of test decks, it may fail in certain data situations that have yet to be encountered. Because the i/o portions of the code are survey-specific, they are very difficult to port to new surveys because the size and initial values of several of the data structures need to be hardcoded in the include files.

Strengths

The DISCRETE system deals with completely general edits of discrete data. If the FORTRAN include files (see above) can be properly changed, then the software is straightforward to apply in all situations. Checking the logical consistency of the set of edits (via gened.for) does not require test data. Error localization (via edit.for) should be far faster than under previously written FH systems for discrete data.

Maintenance of DISCRETE Code

As it is presently written, DISCRETE code is not sufficiently well organized and documented so that it can be maintained. Hundreds of lines of code associated with i/o and data structures are survey-specific.

Future Work on DISCRETE

The DISCRETE system will be improved with general i/o modules, more efficient algorithms for determining the acceptable value-states of the set of error-localized variables, and an indexing method for tracking the set of imputes for each set of edit failures. The optimization loops of the error-localization code may also be improved. The advantage of the indexing method is that it will make the code more easily useable on large surveys such as censuses because many of the optimization loops associated with error localization will only be used once. A loop in the future code will produce a string based on the set of failing edits, perform a binary tree search on previously computed strings associated with edit failures, find the index and set of error-localized fields if the index exists, and, if the index does not exist in the existing trable, perform optimization and add the appropriate error-localized fields for the new index. The main overhead of the indexing method is a sorting algorithm that periodically rebalances the binary tree after a certain number of updates.

Example

The example basically shows what the inputs and outputs from running the two programs of the DISCRETE system look like. The first program generates all the implicit edits that are needed for error localization and checks the logical consistency of the entire edit system. An edit system is inconsistent when no data records can satisfy all edits. The second program uses the entire set of implicit edits that are produced by the first program and edits data records. For each edit-failing record, it determines the minimum number of fields (variable values) to change to make the record consistent.

Implicit Edit Generation

The first program, gened.for, takes a set of explicit edits and generates a set of logically derived edits. The edits are generated by the procedure of FH and consist of the smallest set needed for error localization. Two tasks must be performed. The first is to create an input file of explicit edits. The edits are generally created by subject-matter analysts who are familiar with the survey. An example is given in Table 1. There are 5 edits involving 6 fields (variables). The kth variables takes values 1, ..., nk, where the number of values nk must be coded in a parameter file. A record fails the first edit if variable 1 takes values 1 or 2, variable 4 takes values 1 or 2, and variable 5 takes value 1. Variables 2 and 3 may take any values in edit 1.

Table 1.--Example of Explicit Edit Input File

```
Explicit edit #  1:  3 entering field(s)
     VAR1          2 response(s):  1  2
     VAR4          2 response(s):  1  2
     VAR5          1 response(s):  1

Explicit edit #  2:  4 entering field(s)
     VAR2          2 response(s):  3  4
     VAR3          1 response(s):  2
     VAR5          1 response(s):  2
     VAR6          2 response(s):  1  2

Explicit edit #  3:  3 entering field(s)
     VAR3          1 response(s):  1
     VAR4          2 response(s):  2  3
     VAR6          3 response(s):  2  3  4

Explicit edit #  4:  2 entering field(s)
     VAR2          2 response(s):  1  2
     VAR4          2 response(s):  1  3

Explicit edit #  5:  3 entering field(s)
     VAR1          2 response(s):  2  3
     VAR3          1 response(s):  2
     VAR6          1 response(s):  1
```

Table 2.--Example of Selected Implicit Edits from Output File

6	VAR3	VAR4	VAR5	VAR6
	1	0	0	0
		1		
7	VAR3	VAR4	VAR5	VAR6
	1	0	1	0
		2		1
8	VAR4	VAR5	VAR6	
	2	1	1	
9	VAR3	VAR4	VAR6	
	1	0	0	
10	VAR2	VAR4	VAR5	VAR6
	2	1	1	1
	3	2		
11	VAR2	VAR3	VAR6	
	0	0	1	
	1		2	
			3	

The second task is to change a parameter statement at the beginning of the program and recompile the program. The statement has the form

PARAMETER (MXEDS=20,MXSIZE=8,NDATPT=8,NEXP=5,NFLDS=6).

MXEDS is the upper bound on the storage for the number of edits. MXSIZE is the maximum number of values that any variable can assume. NDATPT is the sum of the number of values that all the variables assume. NEXP is the number of explicit edits. NFLDS is the number of variables (Table 2).

The example of this section is a modified version of the example of GKL. The modification consisting of permuting the variables as follows: 1 -> 3, 2 -> 4, 3 -> 5, 4 -> 6, 5 -> 1, and 6 -> 2. The DISCRETE software generates all 13 implicit edits whereas the GKL software generate 12 of the 13 implicit edits. With an example using actual survey data and 24 explicit edits, the DISCRETE software generates all 7 implicit edits whereas the GKL software generates 6 of 7. The reason that the GKL software does not generate all implicit is due to the manner in which the tree of nodes is traversed. The GKL software traverses the tree of nodes according to their theory.

Error Localization

The main edit program, edit.for, takes two inputs. The first is the set of implicit edits produced by gened.for. The second input is the file being edited. A FORTRAN FORMAT statement that describes the locations of the input variables in the second file must be modified. A large parameter statement that controls the amount of storage needed by the program is not described because of its length. Eventually, the parameter statement will have to be described in comments.

Two output files are produced. The first consists of summary statistics. The second (see Tables 3 and 4) contains details of the edits, blank fields, and possible imputations for each edit-failing record. The edit code presently only delineates acceptable values for the fields designated during error localization. The actual imputed values could be determined via statistical modelling by analysts. The imputation could be written into a subroutine that would be inserted at the end of error localization.

In a typical application, the revised values (Tables 3 and 4) would not be left blank but would be imputed according to rules developed by analysts familiar with the specific set of survey data.

Application

A prototype application of the DISCRETE edit was developed for the New York City Housing and Vacancy Survey (NYC-HVS). This prototype was used to edit ten of the primary fields on the questionnaire. Data collected via the NYC-HVS are used to determine rent control regulations for New York City. The variables that we used in edits were: TENURE, PUBLIC HOUSING?, TYPE OF CONSTRUCTION (TOC), TOC CODE, YEAR MOVED, YEAR BUILT, YEAR ACQUIRED, CO-OP OR CONDO, RENT AMOUNT, and OWNER OCCUPIED. With previous edits, these fields were edited sequentially, starting with the TENURE field. The TENURE field reports whether the occupant of the dwelling is

- the owner,
- pays rent, or
- lives there rent free.

Table 3.--First Example of Edit-Failing Record in Main Output from EDIT.FOR

```
        Record #    1 (    1)       ID: 1001

Implicit edit #  1 failed:
   1. VAR1        :  2
   4. VAR4        :  1
   5. VAR5        :  1

 Implicit edit #  5 failed:
   1. VAR1        :  2
   3. VAR3        :  2
   6. VAR6        :  1

Implicit edit #  6 failed:
   3. VAR3        :  2
   4. VAR4        :  1
   5. VAR5        :  1
   6. VAR6        :  1

    Deleted fields:
   ------------------
   6. VAR6             5. VAR5

The weight of the solution is  2.1100

imputation candidates for field  6. VAR6        :
   3. 3
   4. 4

imputation candidates for field  5. VAR5        :
   2. 2
```

Field names	Reported	Revised	Weights	Failed Edits
VAR1	2. 2	2. 2	1.100	2
VAR2	4. 4	4. 4	1.090	0
VAR3	2. 2	2. 2	1.080	2
VAR4	1. 1	1. 1	1.070	2
*VAR5	1. 1	-1.	1.060	2
*VAR6	1. 1	-1.	1.050	2

Table 4.--Second Example of Edit-Failing Record in Main Output from EDIT.FOR

```
        Record #   2 (   2)        ID: 1002

 Implicit edit #  1 failed:
   1. VAR1       : 2
   4. VAR4       : 1
   5. VAR5       : 1

 Implicit edit #  4 failed:
   2. VAR2       : 1
   4. VAR4       : 1

 Implicit edit #  5 failed:
   1. VAR1       : 2
   3. VAR3       : 2
   6. VAR6       : 1

 Implicit edit #  6 failed:
   3. VAR3       : 2
   4. VAR4       : 1
   5. VAR5       : 1
   6. VAR6       : 1

 Implicit edit #  7 failed:
   2. VAR2       : 1
   3. VAR3       : 2
   5. VAR5       : 1
   6. VAR6       : 1

 Implicit edit # 16 failed:
   1. VAR1       : 2
   2. VAR2       : 1
   5. VAR5       : 1

 Deleted fields:
 ---------------
   5. VAR5          6. VAR6          4. VAR4

   The weight of the solution is   3.1800
 imputation candidates for field  5. VAR5          :
   2. 2
 imputation candidates for field  6. VAR6          :
   2. 2
   3. 3
   4. 4
 imputation candidates for field  4. VAR4          :
   2. 2

 Field names      Reported     Revised      Weights     Failed Edits
 ---------------  -----------  -----------  ----------  ------------
      VAR1          2. 2         2. 2       1.100          3
      VAR2          1. 1         1. 1       1.090          3
      VAR3          2. 2         2. 2       1.080          3
     *VAR4          1. 1        -1.         1.070          3
     *VAR5          1. 1        -1.         1.060          4
     *VAR6          1. 1        -1.         1.050          3
```

A sequential edit, implies an edit based on if-then-else rules. The advantage of sequential edits is that they are often easily implemented. A principal disadvantage is that they are not easily checked for logical consistency. Another disadvantage is that there has to be a initial field from which the remaining fields will be edited. The TENURE field was the initial field for the Annual Housing Survey (AHS). The initial field is never edited in the sequential edit application but can be using a Fellegi-Holt model.

The prototype edit considers all fields simultaneously. The TENURE field was edited in the same manner as the other nine fields. It would be a correct assumption that most respondents are aware of their living arrangement, making TENURE a very reliably reported field. Therefore, TENURE did hold a higher weight. However, there are other circumstances that would cause it to be incorrect. It still needed to be edited.

The explicit edits needed for the DISCRETE prototype were developed from the edits of the prior set of sequential edits. Only the 24 edits that exclusively included the ten fields were considered. Because of the existing sequential edits, the explicit edits needed for the prototype DISCRETE edit were developed with very minimal support from the subject-matter specialists. These 24 edits were run through the edit generator, gened.for, and 8 implicit edits were computed. The edit generator reduced the number of explicit edits to 23, because it determined that one of the explicit edits was redundant. There was now a total of 31 edits for the ten data items.

The DISCRETE prototype produced edited data that were only slightly cleaner than the sequential edit because the data for the AHS were quite clean. The AHS is a long-term survey in which responses are obtained by experienced enumerators rather than via mail responses. The results of the prototype edit were similar to those of the previous sequential edit, except for one striking difference. Using the prototype edit, the TENURE field was in conflict with other fields more often than the subject-matter staff had anticipated.

A second prototype was developed for the Survey of Work Experience of Young Women. This prototype showed the power of the DISCRETE system because it allowed the editing of a large number of data items involving a very complicated skip pattern. No edits had previously been developed for these items because of the complicated nature of the edit situation. The core data items consisted of WORKING STATUS, HOURS/WEEK, HOURS/WEEK CHECK-ITEM, OFFTIME, OVERTIME, and CURRENT LABOR FORCE GROUP. Overall, this prototype was developed for 24 data items. Using previous edit systems, these data items were not edited because of their complex relationships and skip patterns. However, these skip patterns were incorporated into the prototype as explicit edits. This turned out to be a surprising advantage of the simultaneous edit. Working with subject-matter staff, 42 explicit edits were developed for the 24 data items. The edit generator computed an additional 40 implicit edits for a total of 82 edits. Because of the use of the method of data collection used for this survey, the data were very clean. However, the results of this prototype were not as important as was the fact that the prototype was able to edit relationships that were previously considered too complex.

Summary

The DISCRETE system is a Fellegi-Holt edit system for general edits of discrete data. It is a prototype system that is written in FORTRAN. As currently written, it is not maintainable and not

easily portable. Due to new theoretical/algorithmic characterizations (Winkler, 1995b), the system should be more generally applicable than any currently existing system. Although no speed tests have been done, the software should be approximately as fast as other currently existing edit systems.

References

Fellegi, I. P. and Holt, D. (1976). A Systematic Approach to Automatic Edit and Imputation, *Journal of the American Statistical Association*, 71, 17-35.

Garfinkel, R. S., Kunnathur, A. S. and Liepins, G. E. (1986). Optimal Imputation of Erroneous Data: Categorical Data, General Edits, *Operations Research*, 34, 744-751.

Winkler, W. E. (1995a). DISCRETE Edit System, computer system and unpublished documentation, Statistical Research Division, U. S. Bureau of the Census, Washington, D.C., USA.

Winkler, W. E. (1995b). Editing Discrete Data, *Proceedings of the Section on Survey Research Methods, American Statistical Association*, to appear. ■

Chapter 2

Generalized Edit and Imputation System for Numeric Data

Joel Bissonnette, Statistics Canada

Abstract

Statistics Canada's Generalized Edit and Imputation System (GEIS) serves the needs of the professional statistician, and can be used to satisfy the edit and imputation requirements of primarily quantitative surveys. For the editing phase, it checks the consistency of edit rules and produces many summary tables based on survey data, which help to determine the edit rules to apply; it also uses advanced linear programming techniques to flag the fields to be imputed, according to the edit rules. Outlier detection is also available.

For the imputation phase, a choice of three methods is offered. Deterministic imputation identifies fields for which there is only one possible value that allows the record to pass the original edits. Donor imputation replaces the values in error by imputing those from the valid nearest-neighbour record that is most similar to the record in error. Imputation estimators provide imputation for individual fields using a variety of estimators: ratio, current and historical means, and historical values with or without trend adjustments. ■

Chapter 2

The New SPEER Edit System

William E. Winkler and Lisa R. Draper,
U.S. Bureau of the Census

Abstract

This paper describes the methodology, the workings, and an application of the SPEER (Structured Programs for Economic Editing and Referrals) edit system.

The original SPEER, developed by Brian Greenberg, is a Fellegi-Holt system for editing ratios of economic data and has been used on some of the largest U.S. economic surveys. The advantages of Fellegi-Holt systems are

- they check the logical consistency of an edit system prior to the receipt of data,
- the main logic resides in reusable mathematical algorithms with control by easily maintained input tables, and
- with one pass against an edit-failing record, the minimal number of fields are changed to values such that the entire record satisfies all edits.

The current SPEER system consists of entirely new FORTRAN source code and computational algorithms, is exceedingly fast, and is portable across all known computer systems. Merely by changing a parameter file consisting of input and output filenames and input FORTRAN format statements, SPEER can be run against entirely different surveys.

The New SPEER Edit System

William E. Winkler and Lisa R. Draper
U. S. Bureau of the Census

Description of the SPEER Edit System

The following subsections describe aspects of the SPEER edit system.

Purpose, Model, and History

The SPEER edit system is designed for ratio edits of continuous economic data. The system utilizes the Fellegi-Holt model of editing. The first version of SPEER was written by Brian Greenberg (Greenberg and Surdi, 1984; Greenberg and Petkunas, 1990) and the current version was written by William Winkler (1995). The computational algorithms, much of the imputation methodology, and the source code in the current version is new.

Software and Computer Systems

The software consists of two programs, gb3.for and spr3.for. The software is written in portable FORTRAN which should recompile on a variety of computers. It currently runs on IBM PCs under DOS, Windows, or OS/2, DEC VAXes under VMS, DEC Alpha under Windows NT, UNISYS, and a variety of UNIX workstations. The programs run in batch mode and the interface is character-based.

The first program, gb3.for, generates the entire set of edit bounds. The main input is a file containing at least three lines. The first line is the name of the input file of explicit edits, the second is the name of the output file of implicit edits, and the third is the name of the output summary file. If a fourth line is present, it consists of the FORTRAN FORMAT for the input file of explicit edits. If a file of variable names, BNAMES.DAT, is present, then the variable names in it are used; otherwise, default names of the form VRnnn are used where nnn can range as high as 999. After the main input is read, the inputs and outputs are the usual ones associated with edit-generation programs. The most important input is the file of explicit edits that have been defined by an analyst. This input must be in a fixed format that is specified in the program documentation. As output, gb3.for produces the file of implicit edits that are logically derived from the explicit edits and also checks the logical consistency of the entire set of edits. With appropriate test data, an auxilliary program D-MASO (also in FORTRAN) can help an analyst determine the lower and upper bounds on the ratios that are in the set of explicit edits. The appropriate test data might consist of prior year's edited data or (a subset of) the current year's data.

The second program, spr3.for, performs error localization (i.e., determines the minimal number of fields to impute for a record failing edits) and then does imputation. The main input is a control file with at least five lines. The five lines are (1) the name of the input file being edited, (2) the name of the file containing implicit edits, (3) the name of the output, (4) the FORTRAN format of the quantitative data in the file being edited, and (5) the number of variables (fields) being edited. Five additional lines are also

read in. They are (6) the name of the file containing implicit edits, (7) the name of the file containing variable names, (8) the name of the file of beta coefficients, (9) the file of weights, and (10) optional FORTRAN FORMAT for file of explicit edits. The first six lines are mandatory. If the last five lines or the associated files do not exist, then defaults are used. The weights affect which fields are imputed. The variables with lower weights are imputed before those with higher weights. After the control file is read, the input files consist of the set of implicit edits produced by gb3.for, the data file being edited, and a set of "beta" values associated with ratios. The beta values are determined a priori using an appropriate test deck and consist of regression coefficients under the model $y = \beta x$. There can be as many coefficients as there are implicit edits. The imputation methodology consists of first determining an imputation range for a variable so that edits are satisfied. Within the range, the first choice of imputation uses a reported variable that is not being imputed and the corresponding "beta" coefficient. After the first choice, a hierarchy of defaults based on the imputation range is selected. Regression imputation is only used when the appropriate beta coefficient is available and the variable being imputed is associated with a variable that is reported. By the Fellegi-Holt theory, any values of fields chosen in the imputation range necessarily yield complete multivariate records that satisfy all edits.

The outputs from the second program consist of summary statistics, the file of edited (i.e., containing imputes) data, and a file giving details of each record that was changed. The details consists of the failed edits, the minimum fields to impute, and the imputation methodology that was utilized for each field.

Documentation

Three documents describe the overall SPEER methodology and capabilities. They are Greenberg and Surdi (1994), Greenberg and Petkunas (1990), and Greenberg, Draper, and Petkunas (1990). The documents do not describe details of the algorithms or how to create and run the system for specific data bases. New computational algorithms (Winkler, 1995) eliminate much of the redundant computation of earlier versions of the SPEER system. Major restructuring of the computer code makes the system much easier to apply in new situations because only one FORTRAN FORMAT statement describing locations of input fields in the file being edited must be changed. Winkler (1994) describes how to develop and run a SPEER system.

Documentation related to the details of the software and how to run the software has been created for the first time (Winkler, 1995). The main documentation consists of instructions on how to run the example that is included on the disk with the software. Each program has internal documentation (in comments at the end) describing the nature and structure of the inputs and the outputs. The internal documentation should be sufficient to allow all but the most naive users to apply the software in a variety of situations. The new source code is more easily understood because of its modular structure. In most applications it is unlikely that source code (with the possible exception of two parameters that determine that amount of allocated storage) will need modification.

Limitations

SPEER only deals with ratio edits. For a new user, the file of explicit edits may not be very easy to develop. A statistical package should be used to determine those variables that are linearly related and the associated regression ("beta") coefficients. The regression model is $y = \beta x$. Those "beta" coefficients that are placed in an external file are used for the default imputations. If "beta" coefficients are not available for two variables that are associated via a ratio edit, then the default imputation is based on

allowable range that satisfies the edits. The best imputations require survey-specific modifications in which the imputation module is replaced by special code.

The main output from spr3.for is a large print file that contains details of the failed edits, the error localization, and the imputations that were made. The program spr3.for does not produce an output file that has the same FORMAT as the main input file being edited and that has appropriate quantitative data (missing or edit-deletes) replaced by imputations. This is not done due to the difficulty in writing necessary generalized i/o routines, documenting the routines, and getting users to understand how to carry and output additional information from the input file that does not pass through SPEER edits. The program spr3.for does produce an output file EDIT.OUT that contains all the quantitative data fields that pass through the edits and that contains the newly imputed values. It is output in a fixed format and could be merged in with the original data that passes through the edits because it corresponds on a line-by-line basis.

The program spr3.for does not impute values for variables in connected sets in which all values are blank. A set of variables is connected if they are connected via ratio edits. Connected sets form a natural partition of the entire set of variables being edited. If all variables in a connected set are missing, then imputation cannot be based on ratios and must be determined via default procedures that might possibly be based on data from a prior time period.

Strengths

The software is very easy to apply because only one format statement describing the locations and sizes of the quantitative being edited needs to be changed (Winkler, 1995). In situations where storage does not exceed the default storage of the program, the FORTRAN format statement can be read in from an external file. Thus, the software does not need to be recompiled when it is used on different data files. While the software will handle a moderately large number of variables (200+), the present computational algorithms, with suitable modification, could allow it to handle more than 2,000 variables. The software is fast. For instance, to generate 272 pairs of implicit edit bounds in each of 546 industrial categories for the Census of Manufactures requires only 35 seconds on a Sparcstation 20. Because ratio edits are basically simple, algorithms and associated source code are quite straightforward to follow or modify. For most situations, source code should not need any maintenance or modification. All core edit algorithms are in debugged code that is reusable. Checking the logical consistency of the set of edits (via gb3.for) does not require test data. Default imputations are quite straightforward to set up. A new software program cmpbeta3.for will compute the "beta" coefficients for all pairs of variables (fields) that are associated via the ratio edits that are explicitly defined. The program cmpbeta3 is approximately 50 times as fast as commercial software because it contains no diagnostics or special features.

Developing and Running A SPEER Edit System

This section provides an overview of how to create and run the SPEER edit system. It describes some of the non-SPEER components that must be used in addition to the SPEER components. It also gives the type of personnel that are useful as an edit team developing a system.

Developing an Edit System Using SPEER

There are three facets to the development:

- ❒ analysis of the data using statistical and other packages,
- ❒ development of a pre-edit system, and
- ❒ development of a SPEER system. If data from a prior time period are not available, then data obtained during the collection can be used.

Stage 1 proceeds with a variety of steps. The analyst would begin by running various tabulations on the data to determine means, variances, ranges, and other values. Next the analyst would run a regression package to determine which continuous variables are linearly related and to get a variety of diagnostics. The pairs of variables that are linearly related and the associated "beta" coefficients from the regression need to be stored. When data from a prior time period is available, then analysts often have much of this information already.

Stage 2 consists of preliminary edits that often do not require sophisticated rules. These can involve checking whether a State code takes a value within a set of correct values, a variable takes a value in a specified range, and a group of variables adds to a desired sum.

Stage 3 begins with determining the edit bounds for ratios. To facilitate the process, we have a software tool, D-MASO, developed by David Paletz, that delineates potential bounds and a variety of diagnostics. The analysts can then quickly determine bounds. SPEER software consists of two components. The first, gb3.for, generates the logically implied edit bounds and checks the consistency of the entire edit system. It does not require test data. The second component consists of the SPEER edit, spr3.for. It determines edit failures, the minimum number of fields (variables) that must be changed so that the record satisfies edits, and then does imputation. The first program only needs the set of explicit edit bounds as input. The second program needs the set of implicit edit bounds from the first program, the set of "beta" coefficients from the regressions, and the data file that is being edited. A new program cmpbeta3.for will compute the "beta" coefficients for all pairs of variables that are connected via ratio edits. The program requires the file of explicit ratio edits, the main file being edited, the FORTRAN format of the quantitative data in the file being edited, and the number of variables (fields) being edited. It computes beta coefficients for all pairs of variables that can be associated via implicit ratio edits.

Maintenance of SPEER Code

The code may not require any maintenance. If larger data structures are needed, then the two parameters at the beginning of the code should be changed and the program recompiled. If the imputation module is changed or a new one is developed, then updating merely involves substituting the new subroutine for the old.

The code is very modular and contains much internal documentation. In particular, comments at the end of the code give details related to running the programs.

Other Maintenance of a SPEER System

The analyst must document how the "beta" coefficients from the regressions are obtained. The program cmpbeta3.for can quickly produce the set of "beta" coefficients.

Edit Team

An edit team is most useful when it consists of at least one individual in each of the following categories: methodologist, analyst, and programmer. Development of an edit system is primarily a programming project once subject-matter and analytic needs are identified. The methodologist could be an economist, demographer, or statistician who is familiar with the Fellegi-Holt theory and can facilitate the programming of the system. The methodologist can provide an important focal point if the methodologist can make sure that programmers are given knowledgable information about system requirements and understands details of programming such as how long it takes programmers to develop new difficult skills. The analyst is a subject-matter specialist who is familiar with the industries for which data are being edited. Often analysts and programmers have worked together successfully on other projects. Teams often start slowly because of the time needed to develop common terminology and communication skills. Once team members are working closely together, however, final products are often better because individuals are stimulated by detailed knowledge provided by other team members.

Example

The example basically shows what the inputs and outputs from running the two programs of the SPEER system look like. The first program generates all the implied edits that are needed for error localization and checks the logical consistency of the entire edit system. An edit system is inconsistent when no data records can satisfy all edits. The second program uses the entire set of edits that are produced by the first program and edits data records. For each edit-failing record, it determines the minimum number of fields (variable values) to change to make the record consistent.

Implicit Edit Generation

The first program, gb3.for, takes a set of explicit edits and generates a set of logically derived edits. The edits consist of the lower and upper bounds on the ratios of the pairs of variables. Two tasks must be performed. The first is to create an input file of explicit ratio bounds. The bounds are generally created by subject-matter analysts who are familiar with the survey. An example is given in Table 1. The eight fields of the input file are: form number, edit-within-form-number, variable number of numerator, variable number of denominator, lower bound on ratio, upper bound on ratio, an intermediate value between the lower and upper bounds, and the four-character names of the variables. The form number describes the industry to which the edit refers. With U.S. Bureau of the Census surveys, the same form may be sent to all companies over a broad range of industrial classification categories. Separate ratio bounds need to be developed for each industrial classification.

Table 1.--Example of Explicit Ratio Bound Input File

110	1	1	2	.0212400	.0711125	.0369900	EMP1/APR2
110	2	2	3	1.5369120	6.8853623	3.2590401	APR2/QPR3
110	3	3	2	.1670480	.5273000	.3068400	QPR3/APR2
110	4	4	2	.0202880	.2717625	.0929800	FBR4/APR3

The second field refers to the edit number. It is primarily for the benefit of the analysts and is not used by gb3.for. The next two fields are the variable numbers of the fields in the ratio and the following two are the lower and upper bounds created by the analysts. The final two fields are not used by gb3.for but can be used by the analyst. The next-to-last field is possibly an average or median value that the analyst enters in the input file. The last field is a character representation that helps the analyst remember the variables. For instance, QPR3 might refer to "quarterly payroll" and APR2 might refer to "annual payroll."

The second task is only needed if default storage allocations are not sufficient. The task requires changing a parameter statement at the beginning of the program and recompiling the program. The statement has the form

```
PARAMETER (BFLD=45).
```

BFLD refers to the upper bound on the number of variables (here 45) being ratio edited. The number of variables being edited is assumed to be the same in every industry if more than one industry is edited. For the example, the output file primarily contains the ratio bounds (implicit edits) for the six pairs of the four variables.

Error Localization

The main edit program, spr3.for, takes three inputs. The first is the set of implicit edit ratios produced by gb3.for. The second is a set of "beta" coefficients that are created by a regression package that the analyst has used. The third input is the file being edited. A FORTRAN FORMAT statement that describes the locations of the input variables in the third file must be modified and placed in an external file. A parameter statement at the beginning of the program

```
PARAMETER(BFLD=45,BCAT=3,NCENVL=BFLD,NFLAGS=9,N_FLG=100,
 + NEDIT=BFLD*(BFLD-1)/2,MATSIZ=BFLD)
```

must also be changed. BFLD and BCAT are upper bounds on the amount of storage that is allocated. NFLAGS and N_FLG are upper bounds on storage for errors for a single record. In many situations, the default values of these parameters will be sufficient. If they are not, then parameter values will need to be increased and the program must be recompiled. Comments at the end of the source code give many details of setting up and running the program.

Two output files are produced. The first consists of summary statistics. The second (see Table 2) contains details of the edits, blank fields, and imputations for each edit-failing record. The output shows what edit has failed, the minimum number of fields that must be imputed, the imputation method that was adopted, and the revised and reported values of the record.

The program spr3.for is set up so that a more sophisticated imputation can easily be substituted for the existing one. Basically, analysts would have to do more modelling and determine a hierarchy of imputations that would be coded in a subroutine. The imputation subroutine would be added to the code and the eight lines associated with the existing (default) imputation would be replaced by a call to the subroutine. Documentation in the code clearly shows where the substitution should be made and what data must be passed to and from the imputation subroutine.

Table 2.--Example of Edit-Failing Record in Main Output from SPR3.FOR

```
Record #  1
    Failed edits:
        1.8964540  < APR2 / QPR3  <    5.9863030

Deleted fields:   3. QPR3
Imputation range for  QPR3 :  Lo =       3.3410      Up =  10.5460
  QPR3  imputed using  QPR3 / EMP1  ratio

          Fields       Revised      Reported      Lower       Upper
          -------      ----------   ----------    --------    --------
          EMP1          1.000        1.000         .425        1.422
          APR2         20.000       20.000       14.062       34.207
          QPR3          5.714       13.000        3.341       10.546
          FBR4          3.000        3.000         .406        5.435

Record #  5
    Failed edits:
       .0402807 <  EMP1 / QPR3 <      .4257010
      1.8964540 <  APR2 / QPR3 <     5.9863030

Deleted fields:   3. QPR3
Imputation range for  QPR3 :  Lo =       6.6819      Up =  21.0920
  QPR3  imputed using  QPR3 / EMP1  ratio

          Fields       Revised      Reported      Lower       Upper
          -------      ----------   ----------    -----       -----
          EMP1          2.000        2.000         .850        2.845
          APR2         40.000       40.000       28.124       68.415
```

Application

SPEER is currently being used in two large interactive applications. These applications are the Annual Survey of Manufactures and the Census of Manufactures and Mineral Industries. The applied system, named LRPIES (Late Receipts Processing and Interactive Edit System), is used primarily for basic data entry and editing, editing of late receipts, and processing establishment adds. The current version has features that facilitate analysts' review and correction of data records. Analysts in Washington can now enter and correct late receipts that arrive after the central data processing center in Jeffersonville, Indiana has shut down. Previously, late data were entered but generally left unedited. Analysts can also perform additonal review of the non-late data that were previously edited at the Jeffersonville location.

The SPEER application (LRPIES) involves the largest U.S. surveys of industry and manufacturing. As much analyst review of data is needed, custom software modifications that provide assistance and review capability have been added. The modifications are specific to Digital VAXes and the large

screen display capabilities of the types of VAX terminals in use. Records that have failed edits and that require imputation to make them consistent with the set of edits can be retrieved and processed interactively. For each edit-failing record, a number of values are displayed that facilitate the analysts' review and correction. The values are current values, a prior time period's corresponding values if available, suggested impute values, and ranges in which values can be imputed that are consistent with the set of edits. Analysts -- possibly after a call-back -- have the capability of entering a flag that causes an edit-failing value to be accepted. The custom code in LRPIES associated with the interactive edits is the majority of the code. The main SPEER subroutines merely need to be called and do not need to be modified.

The LRPIES application needs edit parameters and information for 546 SIC (Standard Industrial Classification) codes. The main edit parameters are the lower and upper bounds associated with the ratios being edited. Bounds from a prior year are often used as the starting point in producing the bounds for the current year's edits. Edit bounds and information can vary substantially across SIC codes. The specific parameters and information are the implicit edits for the current year and the prior year, the industry average value, and the beta coefficients obtained from regressing one of the variables (fields) in a ratio against the other variable. While the basic SPEER imputation merely uses a regression imputation, the LRPIES application uses a hierarchy of imputations based on the existence of prior data. The exact types of imputations and the hierarchy are determined by analysts familiar with the data.

Summary

The SPEER system is a Fellegi-Holt edit system for ratios of linearly related data. It is written in portable FORTRAN, easily applied, and very fast. Applications of SPEER include some of the largest U. S. economic surveys.

References

Greenberg, B. G.; Draper, Lisa; and Petkunas, Thomas (1990). "On-Line Capabilities of SPEER," presented at the Statistics Canada Symposium.

Greenberg, B. G. and Surdi, Rita (1984). "A Flexible and Interactive Edit and Imputation System for Ratio Edits," SRD report RR-84/18, U.S. Bureau of the Census, Washington, D.C., USA.

Greenberg, B. G. and Petkunas, Thomas (1990). "Overview of the SPEER System," SRD report RR-90/15, U.S. Bureau of the Census, Washington, D.C., USA.

Winkler, W. E. (1994). "How to Develop and Run a SPEER Edit System," unpublished document, Statistical Research Division, U.S. Bureau of the Census, Washington, D.C., USA.

Winkler, W. E. (1995). "SPEER Edit System," computer system and unpublished documentation,Statistical Research Division, U.S. Bureau of the Census, Washington, D.C., USA. ■