THE DISCRETE EDIT SYSTEM

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

ABSTRACT

This document describes the DISCRETE prototype edit system. The system is based on the Fellegi-Holt model and applies to general editing of discrete data.

Keywords: set covering, error localization, integer programming, imputation

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.

I. **DESCRIPTION OF THE DISCRETE EDIT SYSTEM**

The following subsections describe aspects of the DISCRETE edit system.

1. **Purpose, Model, and History**

The DISCRETE edit system is designed for general edits of discrete data. The system utilizes the Fellegi-Holt 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).

2. **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.

3. **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.

4. **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.

5. **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.

6. **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.

7. **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.

II. **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.

1. **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, ..., n_k, where the number of values n_k 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
```

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. 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 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.

2. **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.

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

                                                                             Failed
 Field names       Reported            Revised             Weights   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

                                                                          Failed
 Field names        Reported             Revised              Weights   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
```

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.

III. **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 1) the owner, 2) pays rent, or 3) lives there rent free. 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 AHS Survey. 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, OFF TIME, 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.

IV. **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," *American Statistical Association, Proceedings of the Section on Survey Research Methods*, 108-113.