SET-COVERING AND EDITING DISCRETE DATA

William E. Winkler*, bwinkler@census.gov
Bureau of the Census, Washington DC 20233-9100

Keywords: integer programming, set covering, optimization

This paper describes new set covering algorithms associated with the DISCRETE edit system. DISCRETE is based on the Fellegi-Holt model (JASA 1976) of editing. A new implicit-edit generation algorithm replaces algorithms of Garfinkel, Kunnathur, and Liepins (Operations Research 1986) and Winkler (1995). The new set-covering algorithms correctly gnerate implicit edits for large subclasses and reduce computation during implicit-edit generation by as much as two orders of magnitude.

1. INTRODUCTION

To deal with logical inconsistencies or incorrect data in computer files, we need efficient ways of developing statistical data edit systems that minimize development time, eliminate most errors in code development, and greatly reduce the need to human intervention for manually changing (correcting) records. Fellegi and Holt (1976), hereafter FH, provided the theoretical basis of such a system. Their methods have the virtues that, in one pass through the data, an edit-failing record can be assured to satisfy all edits and that the logical consistency of the entire set of edits can be checked prior to the receipt of data. The implementations of the system have had additional advantages over traditional if-then-else rule edit systems because edits reside in easily modified tables and computer code needs no modification.

Moving Fellegi-Holt principles into survey practice has been slow because of the need to develop sophisticated software algorithms for integer programming and set covering. Early implementations have shown much promise and flexibility. Because all the edits are contained in straightforward tables, sets of edits can be developed rapidly by analysts such as statisticians or economists. If the computer software is well organized, then programmers are not needed. Very large applications consisting of 300 or more edits have been severely hampered by the need for faster hardware and for the development of possibly faster algorithms in operations research. With a work-force survey or a Census long form, the number of edits can range as high as 600.

The key to the FH approach is understanding the *error localization* problem, which is the determination of the (possibly minimal) set of fields to change so that an edit-failing record is altered to one satisfying all edits. In the FH model, a subset of the edits that can be logically derived from the explicitly defined edits (called *implied or implicit edits*) is needed if the error localization problem is to be solved. FH provided an inductive, existence-type proof to their Theorem 1 that demonstrated that is possible to find the region in which the error localization problem could be solved. Their solution, however, did not deal with many of the practical computational aspects of the problem which, in the case of discrete data, were considered by Garfinkel, Kunnathur, and Liepins (1986), hereafter GKL and by Winkler (1995). Because both the implicit-edit generation and the error localization problem are NP-complete (GKL), reducing computation has been the most important aspect in developing a FH-based edit system. Faster algorithms mean that much larger surveys can be processed efficiently.

This paper's main result is an edit-generation algorithm, called the EGE algorithm. It is a much faster alternative to Algorithm 1 of GKL and the EG algorithm of Winkler (1995). For data situations in which no skip patterns are present in the survey form and the edits, the EGE algorithm appears to generate all implicit edits. For other situations, it generates most of the implicit edits. The main present virtue of the EGE algorithm is that it demonstrates that it is possible to generate implicit edits much faster (possibly by several orders of magnitude) in very large data situations. These situations were previously considered computational intractable.

The outline of this paper is as follows. In the second section, we give notation and background material that describe edit generation via set covering algorithms. The third section presents the EGE algorithm for generating implicit edits and some of the heuristics that are associated with it. In the fourth section, we provide some empirical results from a computer system (Winkler 1997) that is based on the new theory and algorithms. The fifth section consists of discussion and the final section is a summary.

2. NOTATION AND BACKGROUND

A record $y=(y_1,...,y_n)$ in a computer file can have n fields subject to edits. For discrete edits, y takes values in $\prod Z^n$, the product space of integers. Each field y_i,

i=1,...,n, corresponds to a variable that is coded. For instance, y_1 might take values 1=male and 2=female. y_2 might take values 1=single, 2=divorced, and 3=married. y_3 might correspond to age and take values 0 thru 99 or 1 thru 99. We set R_n equal to the set of values that field y_n can assume and $D = \prod R_n$. For convenience, we always assume that values in a R_n take values 1 thru k_n where the k_n integers are recodes of the k_n value states associated with field y_n. An edit is a set in D. An analyst might specify that being 12 years or younger is incompatible with being married. Then the corresponding edit E^1 would consist of points having $y_2 = 3$, $y_3 \leq 12$, and the remaining y_is taking any values. FH showed that an arbitrary edit E can be expressed as a union of edits E^i of a particular form. Each E^i can be expressed as $\prod E_{in}$ where E_{in} is the set of values assumed by the nth component of the points y_n in edit E^i. This form of E^i is called the *normal form*. If E_{in} is a proper subset of R_n, then field n is said to *enter* edit E^i and edit E^i is *involved* with field n.

We now make two restrictions that can be made without loss of generality in terms of the theory and practical application in software. The first is that every edit E^i has at least two entering fields. If an edit E^i had only one entering field, then one field, say j, would have at least one value-state that would always result in an error regardless of the values that other fields assumed. For instance, if the jth field consisted of a postal code corresponding to a U.S. State, then we would not consider any such codes that assumed invalid values. Such single-field edits are best dealt with by lookup tables associated with pre-edits in the keypunch software. Thus, while State codes can take any value, we restrict the State codes passed to the edit system of this paper to valid ones. These valid State codes may still be used in multi-field edits because different combinations of edits may be associated with different edits in, say, different States of a national agricultural survey. Our second restriction is that, for each n, $R_n = \cup \{E \in E^o \mid E_{in} \neq R_n\}$ where E^o is the original set of explicit edits defined by analysts. If the union were a proper subset of R_n for some n, then any record y with a component y_n in R_n but not in the union would necessarily pass all edits. The first restriction means that we only consider value-states of fields that enter at least one edit and the second that there are no value-states of individual fields that do not enter at least one field in one edit. In practice, these restrictions could easily be checked via straightforward combinatorial routines. This would alleviate tedious, possibly error-prone checking by analysts. The restrictions facilitate our theoretical development but do not affect software development.

The following lemma of FH is the basis of generating edits in the normal form. For the remainder of the paper, we will only consider edits in the normal form because any system of discrete edits can equivalently be expressed in normal form.

Lemma 1. Let $S = \{E^j, j=1,...,k\}$ be an arbitrary set of normal form edits such that for some field l, E_{jl} is a proper subset of R_j. Let E^* be the edit defined by:

$$E_{*i} = \bigcap_j E_{ji} \quad \text{for } i \neq l \tag{2.1a}$$

$$E_{*l} = \bigcup_j E_{jl} \tag{2.1b}$$

If $E_{*i} \neq \varnothing$ for $i \neq l$, then E^* is an implied edit in the normal form.

If a record *r* fails an implied edit E^* of the form given in Lemma 1, then *r* necessarily fails one of the edits used in generating E^*. The set S is called the *contributing* set of edits used in generating edit E^*. Field l is called the *generating field or node* of E^*. Field l necessarily enters each edit involved in the generation procedure of the lemma. If $E_{*l} = R_l$ then edit E^* is called *essentially new*. In the partial ordering of set inclusion, a normal-form edit is said to be *maximal* if it is properly included in no other normal-form edit. A normal form edit is *redundant* if it is properly included in another normal-form edit. The set of explicit edits plus the set of maximal, normal-form edits is called the *complete* set of edits. The set of original explicit edits is denoted by E^o and the set of complete edits is denoted by E^c. FH had originally defined the set of complete edits as the explicit edits plus the set of essentially new, normal-form edits. GKL noted that the proof of FH for the error-localization problem holds for the complete set as defined in this paper. Our definition of complete is the one due to GKL rather than the one due to FH. A set of edits is *consistent* if there is a least one record that fails no edit.

Using notation similar to GKL, we denote the set of edits generated on node i by (i), those generated on node i and then node j by (ij), and so on. The ordering of the indices in the nodes is important. It is invariant under permutation. The set of implicit edits in a node (ijk) will have i, j, and k as nonentering fields. Additional fields may be nonentering. (ij) and (ijk) are successor nodes of node (i). (ij) is the immediate successor of (i). The set of edits used in generating an implied edit will be called its *generating set*. Generating sets are not unique. Nodes of the form (i) are first-level nodes and implicit edits in first-level nodes are first-level implicit edits.

FH (Theorem 2), with clarification by GKL, showed that all maximal normal-form edits can be generated via

the procedure of the lemma. They observed that if one set of edits is a subset of another and if the generation on field j yields essentially new edits, then the edit generated on the larger set is redundant to the one generated on the smaller set. By similar reasoning, it is also possible to show that if one normal-form edit dominates another (in the set inclusion sense), and if the larger edit replaces the smaller in a generating set of edits, then any generated edit would necessarily dominate the edit that would have been obtained if the smaller edit had been used. For the remainder of the paper, we will assume that the set of edits is consistent.

3. IMPLICIT-EDIT GENERATION THEORY

In this section, we present results and an algorithm for generating the set of implicit edits needed for the complete set of edits E^c. The main result, the EGE algorithm, follows from results of FH, GKL, and Winkler (1995) and the characterizations given in this paper. With most real world data that we have encountered that is not associated with skip patterns, the EGE algorithm generates all implicit edits much more rapidly than previous algorithms. In situations where skip patterns are present, we propose a heuristic that can be used to generate implicit edits that are not generated by the basic form of the EGE algorithm.

We begin by providing a general outline of how implicit-edit-generation must proceed in all situations. Edit-generation first uses explicit edits to generate implicit edits at a root node. Some algorithms then proceed to generate implicit edits at all implicit edits at all nonroot nodes that are successors to the initial root node before proceeding to the second root node (and so on). The algorithm in this paper begins by successively computing implicit edits at all root nodes before proceeding to nonroot nodes. At each root node, information about the implicit edits that were generated and the specific explicit edits that were used each implicit edit is placed in a data structure that keeps track of the root node computation. At nonroot nodes, we make use of the information that tells us what sets of explicit edits were successfully used in computing new implicit edits at the root nodes. Our heuristic is that implicit-edit-generation at nonroot nodes can be successfully tracked by the information from the root nodes. The information must be used in a certain way that will be defined later. In other words, the heuristic is that a successful use of a set of explicit edits in generating an implicit edit at a root node gives us detailed information that we can use at nonroot nodes. In this way we reduce computation at nonroot nodes from a all subsets problem involving K explicit and implicit edits that are used in generation to one involving a large number of very small subsets. With large problems, computation can be reduced several orders of magnitude at each nonroot node.

Our method is a heuristic that does not hold when skip patterns or edit-data structures are present that are similar to skip patterns. In the case of skip patterns, important information needed for correct generation at nonroot nodes cannnot be obtained at the root nodes. The way we propose solving this general situation is to run the basic EGE algorithm until completion. We then run an identification step that identifies root node implicit edits and information associated with them that is needed for computation of the remaining implicit edits. This information can be added to the data structures used in the EGE algorithm and a subportion of certain computational paths is then traversed. With two small test cases, the method has been tested and successfully generates all implicit edits. We are in the process of writing software for the general situation.

The main advantage of this approach is that the basic EGE algorithm is far faster than previous algorithms. At a minimum, it allows us to compute all implicit edits when skip patterns are not present in very large edit situations. The proposed follow-on step may require significantly more computation that the basic EGE step. The follow-on step, however, can be decomposed into subcomponents that allow us to evaluate the total amount of computation needed. Each of the subcomponents will have computation of the same order as the basic EGE algorithm that has successfully completed. With previous algorithms there was no way to evaluate how long implicit edit-generation would take. For instance, there was no way to determine if computation would take one extra day or 1000 extra days.

As we observed earlier, if a non-maximal (i.e., redundant) edit E^i is part of a generating set of edits E^g, then the generated implicit edit will be dominated by (redundant to) the implicit edit that is generated by $E^{gm} = E^g \setminus \{E^i\} \cup \{E^j\}$ where E^j is an edit that dominates E^i. Thus, if we are able to restrict edit-generation to subsets containing non-redundant (possibly maximal) edits, then we can reduce computation.

Lemma 2 (Winkler 1995). In generating the complete edits E^c, E^o can be replaced by E^{om}, where each edit in E^{om} is maximal and dominates at least one edit in E^o.

The set E^{om} is said to be *equivalent* to E^o because it generates the same set of maximal implicit edits. The following lemma is due to FH. In practice, maximal implicit edits that can replaced by explicit edits are identified by a heuristic. We run the main algorithm through the generation of implicit edits at 2nd-level nonroot nodes (i.e., nodes of form (ij)). If explicit edits

are overwritten by maximal explicit edits, we restart the edit-generation process with the new set of edits (which we also call explicit). Ultimately, after running a full edit-generation procedure to conclusion, we need to check whether any explicit edits have been overwritten. If they have, then we need to restart the edit-generation process.

Lemma 3. Let E^i be an edit that is generated by set E^g on node j. Let E^{i*} be an edit that is generated by a proper subset of E^g on node j. Then E^{i*} dominates E^i.

With many real world data situations that we have encountered, the EGE algorithm generates all implicit edits. We can also develop an auxiliary algorithm and code to determine when the algorithm of this paper does not generate all implicit edits and then to generate the balance of the implicit edits.

The following is the edit-generation algorithm.

EGE Algorithm:

1. Replace, if necessary, the original set of explicit edits by an equivalent set of maximal explicit edits.
2. Traverse the tree of nodes in all orders.
3. Keep track of all the maximal implicit edits generated at each root node and each set of explicit edits that were used in generating individual maximal implicit edits.
4. At each nonroot node, for each newly implied edit in the immediate predecessor node, collect the set of potentially edit-generating edits to be passed on to the actual edit-generation step for the successor node.
5. Within each nonroot node, for each new implicit edit in the existing node, systematically generate new, maximal implicit edits as follows. For each, combine it with one of the subsets of explicit edits used in generating on the appropriate field at a root node. Look at all the subsets consisting of n-1 explicit edits and the implicit edit where the set of explicit edits has n edits. Successively use all of the appropriate sets of explicit edits.

In many problems, most of the computation takes place at the root nodes. At root nodes, it is necessary to use brute-force algorithms to look at all possible combinations of explicit edits and determine the exceedingly small set of subsets of them that actually generate new maximal implicit edits. At present, there appears to be no way of eliminating most of the wasted computation at root nodes. The problem is still NP-Complete as shown by GKL. At nonroot nodes, for each implicit edit from the predecessor node and in an efficient manner, we successively combine each small subset of explicit edits that were used in generating on the appropriate field. At nonroot nodes, GKL used all subsets of the set all implicit edits and all explicit edits. At nonroot nodes, Winkler (1995) used a set of subsets where each subset consisted of an implicit edit from the predecessor node and the entire set of explicit edits that could be used in generating implicit edits with it. If there are an very large number of nonroot nodes in which new implicit edits can be generated, then overall computation can increase significantly over the root node computation.

4. IMPLICIT-EDIT GENERATION RESULTS

In this section, we directly compare three different algorithms and indicate how to compare them with the ISTAT-IBM system, the fastest known predecessor system (e.g., Barcaroli et al 1997). For convenience, we use the names of the programs to denote the three different sets of algorithms. The three programs, gen_ed3ebk, gen_ed3f.f and gen_ed4.f, each have additional speedups that were developed earlier (Winkler 1995) and some that were developed the current versions of the code. Each traverses root nodes in the same manner and in a manner similar to GKL. Although we do not yet have details of the ISTAT-IBM system, we assume it traverses root and nonroot nodes in a manner similar to the algorithms of this paper. The programs gen_ed3ebk.f and gen_ed3f.f traverse root nodes in a conventional manner like GKL (GKL's code is available). The program gen_ed4.f traverses nonroot nodes according to the EGE algorithm.

The test deck has 252 explicit edits, 33 fields, and 96 value states for the fields. Three fields have 12 value states and the remaining 30 have 2 value states. The upper bound on computation is of the order M N exp(M) where M are the number of explicit edits and N is the number of value states. ISTAT-IBM has a system with more than 550 explicit edits and 400 value states. In converting the ISTAT-IBM edits to the form used by DISCRETE, we drop several explicit edits that can never be used in generating new implicit edits and we combine several value states in situations were the value states are never used separately in explicit edits. The resultant converted set has 531 edits with 442 value states. We compare algorithms by considering the entire sets of explicit edits and selected subsets. The ISTAT-IBM subset numbers (Barcaroli et al 1997) are based on a partitioning method that further reduces computation from merely taking subsets (the method used in this paper). In the following, we take subsets of the 252 to get a feeling for the computational growth rates using the different algorithms and use an additional deck of 1560 edits that has the same 96 value states. Computational rate numbers are based on a Sun

UltraSparc workstation and the largest IBM mainframe.

Table 4.1. CPU times
Subsets of 252 explicit
Versions of Algorithms

# explicit	- computation time (cpu) ---- genedit version --- 3ebk	3f	4
60	0.7 m	1.5 m	0.1 m
160	11.7 m	16.0 m	0.8 m
252	5.5 h	3.3 h	1.8 m
1560*	NA	NA	70 h

NA - not applicable, m-minutes, h-hours.
*/ Number of implicit edits used in checking run of 70 hours.

Table 4.2. CPU times on Different Subsets
Algorithms and Software
IBM-ISTAT vs EGE

# explicit	--- computation time (cpu) -- ISTAT-IBM	EGE */
163	?6-24 hrs	394 sec
290	?12-24 hrs	5700 sec
24	?less than 10 hrs	14 sec
54	?less than 10 hrs	1 sec
531	did not complete in 8 days on largest IBM mainframe	27.5 hr

*/ With the subset of 290 and the entire set of 531, the EGE algorithm only generates a very large subset of the implicit edits.

5. DISCUSSION

In this section we describe differences in the EGE algorithm, the EG algorithm of Winkler (1995), and Algorithm 1 of GKL. We also discuss limitations of the empirical comparison of section 4. The purpose of the discussion is to give insight into the drastically reduced computation that occurs in many real world data situations in which skip patterns do not occur.

5.1. Differences in Algorithms

The main difference between the EGE algorithm and the EG Algorithm of Winkler (1995) is the drastically reduced amount on computation at nonroot nodes. Computation at root nodes in the EG algorithm, the EGE algorithm, and Algorithm 1 of GKL is identical (and still NP-complete) as shown by GKL. In the following we describe the computational differences between the algorithms at nonroot nodes. The only edits (both implicit and explicit) that are considered are those that enter the current field on which generation is taking place. In the EGE algorithm, we successively use maximal implicit edits from the immediate predecessor node and, for each, we successively use the known small subsets of explicit edits that were successful in generating implicit edits at root nodes. In the EG algorithm, we successively used each maximal implicit edit and all associated explicit edits simultaneously. We, thus, consider far fewer subsets of edits in each generating cycle at nonroot nodes with the EGE than we consider with the EG. In Algorithm 1 of GKL, all implicit and explicit edits were considered simultaneously. With the GKL algorithm, far more subsets of edits are considered at each generating cycle than with the EG or EGE algorithms.

The amount of redundant computation in the EG and GKL algorithms increases at an exponential multiple of the amount of computation at the root nodes. With the EGE algorithm, computation at each nonroot node can be a small subset of the computation at the root nodes. Overall computation can still be substantial if there are a significant number of nonroot nodes at which implicit edits must be generated.

5.2. Limitations of Empirical Results

The program gen_ed4.f may contain some minor errors. At present I have only run it with one additional problem of the same computational complexity as the ISTAT-IBM problem. Several minor errors in the code were corrected. There are undoubtedly some errors in the transcription of the ISTAT-IBM edits into the form used by DISCRETE. I do not believe that the transcription errors seriously affect the comparative timing results given in this paper.

5.3. Further Enhancement to the EGE Algorithm

At present, the EGE algorithm can be enhanced by adding an additional step in which root node implicit edits are used in generating further implicit edits and the sets of implicit edits used in generating implicit edits are tracked with the same data structures as in the current code. This should allow generation of implicit edits associated with simple skip patterns and edit situations analogous to skip patterns. It will likely not allow dealing with skip patterns that occur within skip patterns. We have not yet encountered the skip-pattern within-skip-pattern situation with real data.

6. SUMMARY

This paper presents theory and algorithms that facilitate generation of implicit edits for discrete data under the edit model of Fellegi and Holt (1976). The main algorithm is presently valid for situations in which skip patterns are not present. It holds for many survey situations such as most demographic surveys and the decennial census short form. The advantage of the new implicit-edit-generation algorithm that is far faster (1-2 orders of magnitude) than previous algorithms. Combined with previous methods (Winkler 1995) that reduce computation during the error localization phase (1-2 orders of magnitude), the new set of algorithms allows the development of larger Fellegi-Holt edit systems that can be used in practice.

*This paper represents views of the author and are not necessarily those of the Bureau of the Census. The author thanks Dr. Bor-Chung Chen for identifying an error in the proof of Lemma 4 and helpful conversations The author is especially thankful to Dr. Guilio Barcaroli of the Italian National Statistical Institute (ISTAT) for timely advice and for providing large, exceedingly difficult data sets that were used for testing and for debugging the computer code.

REFERENCES

Barcaroli, G., and Venturi, M. (1997), "DAISY (Design, Analysis and Imputation System): Structure, Methodology, and First Applications," in J. Kovar and L. Granquist, (eds.) *Statistical Data Editing, Volume II*, U.N. Economic Commission for Europe, 40-51.

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.

Little, R. A., and Rubin, D. B., (1987), *Statistical Analysis with Missing Data*, John Wiley: New York.

Nemhauser, G. L. and Wolsey, L. A., (1988), *Integer and Combinatorial Optimization*, John Wiley: New York.

Winkler, W.E. (1997), "DISCRETE 97," Undocumented computer system, Statistical Research Division, U.S. Bureau of the Census.