EDITING DISCRETE DATA

William E. Winkler

ABSTRACT

This paper describes theory, computational algorithms, and software associated with the DISCRETE edit system. The prototype DISCRETE edit system is based on the Fellegi-Holt model (*JASA* 1976) of editing. A new implicit-edit generation algorithm replaces an algorithm of Garfinkel, Kunnathur, and Liepins (*Operations Research* 1986). A characterization specific to the edit situation reduces the amount of information needed in the integer programs used for error localization. Even with moderate-size problems, computation during error localization is reduced by two orders of magnitude.

Keywords: integer programming, set covering, optimization

1. INTRODUCTION

Computer files used for administrative or survey purposes may contain large numbers of records, some of which contain logical inconsistencies or incorrect data. Pritzker, Ogus, and Hansen (1965) describe the nature of the problem. Errors can arise because methods of creating records in files are not consistent, because questions are not understood, or because of transcription or coding problems. In many situations, data files are edited using custom software that incorporate rules developed by subject-matter specialists. If the specialists are unable to develop the full logic needed for the edit rules, then the subsequent edit software is in error. If programmers do not properly code the rules, then the software would be in error. Developing software from scratch each time a data base is redesigned is time-consuming and error-prone. It is better to have a system that can describe edit rules in tables that are read and utilized by reusable software modules. The tables could be more easily updated and maintained than complex if-then-else rules in computer code. The software would automatically check the logical validity of the entire system prior to the receipt of data during production processing.

Fellegi and Holt (1976), hereafter FH, provided the theoretical basis of such a system. FH had three goals that we paraphrase:

1. The data in each record should be made to satisfy all edits by changing the fewest possible variables (fields).
2. Imputation rules should derive automatically from edit rules.
3. When imputation is necessary, it should maintain the joint distribution of variables.

The key to the FH approach is understanding the underpinnings of goal 1. Goal 1 is referred as the *error localization* problem. In the FH model, a subset of the edits that can be logically

derived from the explicitly defined edits (called *implied or implicit edits*) are 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. Because the error localization problem is NP-complete (GKL), reducing computation is the most important aspect in implementing a FH-based edit system.

This paper provides two main results. The first is an edit-generation algorithm, called the EG algorithm, that is an alternative to Algorithm 1 of GKL. Like Algorithm 1 of GKL the EG algorithm reduces computation over algorithms based directly on ideas in FH. We provide a slightly modified version of an example due to GKL that shows that the EG algorithm correctly generates all maximal implicit edits whereas Algorithm 1 of GKL does not. GKL defined maximal implicit edits and observed that FH had essentially shown that the set of maximal implicit edits yield a solution to the error-localization problem. Maximal implicit edits will be defined later in this paper. The second result identifies a smaller set of information that is needed for error localization that can reduce computation by two orders of magnitude with moderate size problems. We observe that the only general integer programming method for solving the error-localization problem is branch and bound (Nemhauser and Wolsey 1988, Garfinkel and Nemhauser 1972) and that branch-and-bound computation grows faster than an exponential of the number of edits needed for error localization.

The outline of this paper is as follows. In the second section, we give notation and background material that describe edit generation and error localization. The third section presents an efficient algorithm for generating implicit edits and an algorithm that significantly speeds error localization. In the fourth section, we provide some empirical results from a computer system (Winkler 1995) 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 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 components 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:

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

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

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 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. We do not claim that node generation is invariant under permutation; that is (ij) = (ji) or (ijk) = (ikj). GKL claimed node generation 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.

We can observe that if we were to apply the FH lemma in a straightforward, brute-force fashion, we would ultimately generate all maximal, normal-form edits. The intent of this paper is to characterize the generation process more clearly so that, at each stage, we use only those edits necessary for maximal, normal-form edits. If we do not have this characterization, then as the edit-generation process proceeds, we generate increasingly more redundant edits or make more unsuccessful attempts because the intersection one of the fields associated with a set of generating edits is null. The unneeded extra computation increases at an exponential rate.

Let Ω_K be the subset of E^c that involves only fields 1, 2, ...,K. The following theorem is the main error localization result of FH.

Theorem 1 (FH). If y_i^o, i = 1, 2, ..., K-1, are, respectively, some possible values of the first K-1 fields, and if these values satisfy all edits in Ω_{K-1}, then there exits some value y_K^o such that y_i^o, i = 1, 2, ..., K, satisfies all edits in Ω_K.

By reasoning inductively, we can fill in y_i^o, i = 1, 2, ..., K-1, with values y_i^o, i = K, ..., N, such that y_i^o, i = 1, ..., N, satisfies all edits in E^c. Since the ordering is arbitrary, we can assume that for any subset s and any set of values y_j^o, $j \in s$, that satisfy edits in E^c with entering fields in s, can be completed to a record that satisfies all edits in E^c. If $r = \{y_i^o, i = 1, ..., N\}$ is a record that fails a set of edits E and s is the set of fields that enter the edits in E, then we can find a minimal cardinality subset s_1 of s so that $\{y_j^o, j \notin s_1\}$ can be completed to a record that satisfies all edits. If we consider weights c_i, i = 1, ..., N, then we can find the minimal weighted subset s_1 of s. We observe that E^c is a set of edits that is sufficient for determining the minimal number of fields (i.e., the set s_1) that must be imputed to change (complete) an edit-failing record to one that satisfies all edits. FH (see also GKL) reformulated the error-localization problem as an equivalent integer programming problem (array $A=a_{ij}$; i = 1, ...,K; j = 1, ..., N) in which the rows correspond to failed edits and the columns correspond to the fields. Entry a_{ij} is 1 if field j enters edit i and is 0 otherwise. Array A is called the failed-edit array. It is well known with such integer programming problems (e.g., Nemhauser and Wolsey, p. 125) that the upper bound on computation is proportional to the product 2^K K N. If we can reduce the number of fields N that must be considered, then we will reduce computation. If we can reduce the number K of implicit edits, then we can reduce computation substantially. For instance, if we had 5 explicit

edits on N = 6 fields, 13 maximal implicit edits, and only 4 of the maximal implicit edits were needed for error localization, we would reduce computation in the error-localization subroutine by as much as a factor of 1000.

FH and GKL gave examples showing that the set of maximal implicit edits are needed for error localization when the error-location problem is translated to the equivalent integer programming problem. Their examples each provided similar characterizations. If explicit edits are the only edits used in creating the failed edit array, then the resultant solution of fields could yield changed field values (i.e., a completed record) that satisfy the set of edits that were failed originally and failed some of the explicit edits that were not failed originally. Thus, the implicit edits provide information about edits that are not failed originally and are necessary for solving the error-localization problem. For the remainder of the paper, we will assume that the set of edits is consistent.

3. **THEORETICAL RESULTS**

The theoretical results of this section are given in three parts. In the first subsection, we present results and an algorithm for generating the set of implicit edits needed for the complete of set of edits E^c. The results are a hybrid of results from FH, GKL, and this paper and are intended to replace the implicit edit-generation results of GKL. While the methods of GKL are computationally superior to those of FH, there is a gap in the reasoning of GKL that can cause their error-generation algorithm to fail to generate all maximal implicit edits in some situations. The error generation algorithm of this paper retains much of the computational superiority of the GKL and is closer the original algorithm of FH. The FH algorithm, while correct, can necessitate too much computation for practical use.

In the second part, we provide a characterization that allows us to reduce the amount of information needed for error localization. In most situations, computation is reduced drastically. The third set of results provide a means of tracking error-localized solutions so that computation need not be repeated. In large survey situations such as censuses, the methods can provide a substantial decrease in computation.

3.1. Implicit Edit Generation

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. 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.

Proof. GKL observed that the proof FH Theorem 2 actually shows that the procedure of Lemma 1 generates all maximal implicit edits. If we go through a straight-forward edit generation, we can monitor which generated edits, if any, replace explicit edits. E^{om} is the set of maximal implicit edits that replace explicit edits plus the set of explicit edits that are not replaced or redundant. If we perform the edit-generation procedure again with E^{om} as the starting set of edits, we generate the E^c.

Lemma 2 means that we can replace an original set of explicit edits E^o with the set E^{om}. Any generated edit that replaces an explicit edit will still be called an explicit edit. Because E^{om} can generate fewer redundant edits, computation can be reduced. At present, I have a straightforward combinatorial routine that identifies E^{om} and uses much logic from the main implicit-edit-generation program. Once E^{om} is identified, the program calls the main implicit-edit-generation algorithm. 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 stated without proof.

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.

The lemmas and corollary yield the implicit edit-generation algorithm.

EG 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. At each node, for each newly implied edit, collect the set of potentially edit-generating edits to be passed on to the actual edit-generation step for the successor node.
4. Within each successor node, for each new implicit edit in the existing node, systematically generate new, maximal implicit edits.

There are two main differences between the EG algorithm and the algorithm of GKL. First, with the EG algorithm, tree traversal requires all orderings of nodes of the form (ijk...); with GKL $i < j < k$. In the next section, we show that unrestricted orderings are needed. Second, with Step 3 of the EG algorithm, we restrict the number of edits passed to the edit-generation loop; with the GKL algorithm, there is no restriction and the number of edits can grow at an exponentially higher rate than with the EG algorithm. Computation within each node is still exponential in the number of edits just as it is under the GKL algorithm.

3.2. Error Localization

The overall strategy for reducing computation is to use a three-step pruning procedure to reduce the amount of information needed during error localization. Since the set of fields in the error-localization must be a subset of the fields in the failing explicit edits, we first identify the entering fields from the failing explicit edits. Second, we delineate all failing implicit edits along with their associated entering fields that are a subset of the entering fields from the failing explicit edits. Third, we delete all failing edits that have sets of entering fields that proper supersets of the sets of entering fields of other failing edits. Any cover of the failing edits with a set of entering fields that is a subset of another necessarily covers the larger failing edit.

Those fields and edits remaining after the pruning are passed to the error-localization that consists of either a branch/bound algorithm (code originally written by Kunnathur) or a greedy algorithm (Nemhauser and Wolsey 1988, p. 466).

4. **EMPIRICAL RESULTS**

The main example of this section is a modified version of an example due to GKL (Table 4.1). The difference is that the basic fields are permuted from the order given in GKL. The value states of individual fields go from 1 to 4 rather than 0 to 3. The fields of GKL are permuted in the following manner: 1 -> 3, 2 -> 4, 3 -> 5, 4 -> 6, 5 -> 1, and 6 -> 2.

Table 4.1. A Six-Field Five-Edit Example
Explicit Edits

	Fields					
	1	2	3	4	5	6
E^1 =	{1,2}	R_2	R_3	{1,2}	{1}	R_6
E^2 =	R_1	{3,4}	{2}	R_4	{2}	{1,2}
E^3 =	R_1	R_2	{1}	{2,3}	R_5	{2,3,4}
E^4 =	R_1	{1,2}	R_3	{1,3}	R_5	R_6
E^5 =	{2,3}	R_2	{2}	R_4	R_5	{1}

Table 4.2 provides the set of edits that are the maximal, implicit edits generated by software that follows the EG algorithm of the previous section. All 13 would have been produced using algorithms of GKL if the fields were left in their original order. Edit 9 at node (152) of Table 4.2 is not produced by the algorithm of GKL. Since GKL believed that nodes are permutation invariant, they did not backtrack which can require significantly extra computation. Specifically, under their algorithm, they assume that node (125) equals (152) and that if (125) exists necessarily its predecessor node (12) exists. Since (12) does not exist, there is no way to generate (125) with the GKL algorithm.

The "Gen by" column actually refers to those original edits used in generating the implied edits. In actuality, E^8 is generated from E^6 and E^2 and E^9 is generated from E^8 and E^4.

5. **DISCUSSION**

5.1. Tracking Optimal Solutions

In situations such as a large census, many edit-failure patterns will be repeated numerous times. To save computation, error-localization solutions could be tracked and reused. Very little overhead is involved.

Specifically, the tracking procedure is a follows. For each edit-failing record, the set of failing maximal implicit edits uniquely determines the error-localization solution. Rather than repeat

Table 4.2. Implicit Edits

	Fields							
	1	2	3	4	5	6	Node	Gen by
E^6 =	R_1	R_2	{2}	{1,2}	{1}	{1}	(1)	1,5
E^7 =	R_1	{1,2}	{2}	R_4	{1}	{1}	(14)	1,4,5
E^8 =	R_1	{3,4}	{2}	{1,2}	R_5	{1}	(15)	1,2,5
E^9 =	R_1	R_2	{2}	{1}	R_5	{1}	(152)	1,2,4,5
E^{10} =	R_1	R_2	{2}	{1,3}	{2}	{1,2}	(2)	2,4
E^{11} =	R_1	R_2	R_3	{3}	{2}	{2}	(23)	2,3,4
E^{12} =	{1,2}	R_2	{2}	{1}	R_5	{1,2}	(25)	1,2,4
E^{13} =	R_1	{3,4}	R_3	{2,3}	{2}	{2}	(3)	2,3
E^{14} =	{1,2}	{3,4}	R_3	{2}	R_5	{2}	(35)	1,2,3
E^{15} =	{1,2}	R_2	{1}	R_4	{1}	{2,3,4}	(4)	1,3
E^{16} =	{1,2}	{1,2}	R_3	R_4	{1}	R_6	(4)	1,4
E^{17} =	R_1	{1,2}	{1}	R_4	R_5	{2,3,4}	(4)	3,4
E^{18} =	{1,2}	{3,4}	{2}	{1,2}	R_5	{1,2}	(5)	1,2

computation each time an edit-failure pattern occurs again, we can create a B-tree that is indexed by the 0-1 strings associated with edit-failure patterns. Edit-failure patterns consist of a string with as many character positions as there are maximal implicit edits. Each entry in the B-tree will contain an index that points to the error-localization solution. If an edit-failure pattern is found in the B-tree, then the error-localized solution is used. If it is not found, then the tree is updated. The only overhead is periodically rebalancing the B-tree to assure that searches occur at a binary rate. With a large training set based on previous data, most of the B-tree could be constructed in advance of production editing.

5.2. Computational Algorithms

Lemma 2 is crucial to the results of section 3.2 because it assures that maximal implicit edits are generated in an efficient manner. At present, I have no constructive proof of Lemma 2 that allows an associated computer procedure. Because the set E^{om} exists, I currently produce it by (1) using explicit edits to generate first level implicit edits, (2) replacing any explicit edit with a first-level implicit edit that exceeds it, and (3) repeating steps (1) and (2) until no edits exceeding explicit edits are produced.

5.3. Software

The software presently consists of FORTRAN driver routines that were originally developed for a specific survey system that tested code originally written by Kunnathur and the new FORTRAN code for edit generation and error localization. The general algorithms for edit generation and error localization compile and run on IBM PCs, Unix Workstations, and VMS VAXes. To get the overall code running on different machines, I had to revise several hundred lines associated with i/o and some data structures. I have not yet written general i/o modules.

5.4. Further Empirical Tests

The edit-generation software was tested with two sets of actual survey data. With each, the EG algorithm generated maximal implicit edits that were not generated by Algorithm 1 of GKL.

5.5. Computational Speed

With the above test data, edit-generation on a Sparcstation 20 took 0.1 second. With a larger test deck of actual survey data having 24 explicit edits and 10 variables, edit generation of 7 maximal implicit edits needed 0.4 second. Since the edit-generation code is used to check the logical consistency of a system prior to receipt of data, speed is not as crucial as it is for error localization which takes place during production.

Error-localization was evaluated via three different procedures using a small test deck of 38 edit-failing records corresponding to the example of section 4. The execution times were: (1) 0.25 with the pruning procedure and the greedy algorithm, (2) 0.33 with the pruning procedure and the branch/bound algorithm, and (3) 0.43 with no pruning and the branch/bound algorithm. The pruning and greedy algorithm required less than 0.005 second in procedure (1). Thus, the error localization with procedure (1) was at least a factor of 50 faster than with the procedure (3) with no pruning and the branch/bound algorithm. With larger data situations, the computational differences should be much greater. Using the data of the example, the greedy algorithm always produced the same solution as the the one produced by the branch/bound algorithm. Generally, we would expect the greedy algorithm to produce sub-optimal solutions.

6. **SUMMARY**

This paper presents theory and algorithms that facilitate generation of implicit edits under the

edit model of Fellegi and Holt (1976). It gives a much more precise characterization of error localization that reduces computation and allows the development of general Fellegi-Holt edit systems that can be used in practice.

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.

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

Pritzker, L., Ogus, J. and Hansen, M. H., (1965) "Computer Editing Methods--Some Applications and Results," *Bulletin of the International Statistical Institute*, *Proceedings of the 35th Session*, Belgrade, 395-417.