Journal of Official Statistics, Vol. 19, No. 4, 2003, pp. 383–402

A Fast and Simple Algorithm for Automatic Editing of Mixed Data

Ton de Waal and Ronan Quere[1]

In order to automate the data editing process the so-called error localisation problem, i.e., the problem of identifying the erroneous fields in an erroneous record, has to be solved. A new algorithm for solving the error localisation problem for mixed data, i.e., a combination of continuous and categorical data, has recently been developed. This algorithm is based on constructing a binary tree, and subsequently searching this tree for optimal solutions to the error localisation problem. In the present article we provide a mathematical description of the algorithm, and prove that the algorithm determines all optimal solutions to the error localisation problem. We also provide computational results for several realistic data sets involving only numerical data.

Key words: Branch-and-bound; data editing; Fellegi-Holt method; Fellegi-Holt paradigm; Fourier-Motzkin elimination.

1. Introduction

Traditionally, statistical agencies have put a lot of effort and resources into data editing activities. They have considered it a prerequisite for publishing accurate statistics. In traditional survey processing, data editing was mainly an interactive activity designed to correct all data in every detail. Detected errors or inconsistencies were reported and explained on a computer screen. Clerks corrected the errors by consulting the form, or by contacting the supplier of the information. It has long been recognised, however, that it is not necessary to correct all data in every detail. Several studies (Granquist 1995, 1997; Granquist and Kovar 1997) have shown that generally not all errors have to be removed from a data set in order to obtain reliable publication figures. It suffices to remove only the most influential errors. These studies have been confirmed by many years of practical experience at many different statistical offices.

Of course, it is true that for a data set to be suitable for statistical analysis the significant errors in the data have to be corrected. However, it is not necessary that each record, i.e., the data of an individual respondent, should be absolutely correct. Statistical offices publish aggregate data, often based on samples of the population. This implies that small errors in individual records are acceptable. First, because small errors in individual records

[1] Statistics Netherlands, Division of Technology and Facilities, Methods and Informatics Department, PO Box 4000, 2270 JM Voorburg, The Netherlands. Email: twal@cbs.nl

Acknowledgment: The views expressed in this article are those of the authors and do not necessarily reflect the policies of Statistics Netherlands. The authors would like to thank the Associate Editor and three anonymous referees for their useful comments.

tend to cancel out when aggregated. This is reported in the literature (see e.g., Granquist 1995), and is confirmed by our own experiences at Statistics Netherlands. Second, because if the data have been obtained by means of a sample from the population there will always be a sampling error in the results, even when all collected data are completely correct. In this case an error in the results due to incorrect data is acceptable as long as this error is small in comparison to the sampling error.

In our opinion, the ideal edit strategy is a combination of selective (or significance) editing (cf., Lawrence and McKenzie 2000; Hedlin 2003), automatic editing, and macro-editing (cf., Granquist 1990). After data entry, simple checks and simple automatic corrections should be applied. Examples of simple checks are range checks. Examples of records to which simple corrections can be applied are cases in which it is clear that a respondent filled in a financial figure in Euros instead of the requested thousands of Euros. After that phase selective editing should be applied to split the data into two streams: the critical stream and the noncritical stream. The critical stream consists of those records that are the most likely ones to contain influential errors; the noncritical stream consists of records that are unlikely to contain influential errors. The records in the critical stream are edited in a traditional, manual manner. The records in the noncritical stream are either not edited or are edited automatically. In practice, the number of records edited manually often depends directly on the available time and resources. Macro-editing, i.e., verifying whether figures to be published seem plausible, is an important final step in the editing process. Macro-editing can reveal errors that would go unnoticed with selective editing or automatic editing.

Editing the records in the noncritical stream automatically has our preference over not editing these records at all. The sum of the errors in the noncritical records may have an influential effect on the publication figures, even though each error itself may be non-influential. Moreover, many noncritical records will be internally inconsistent if they are not edited, which may lead to problems when publication figures are calculated. Automatic editing helps to reduce the errors in the data, and makes sure that the records become internally consistent.

More than a quarter of a century ago Fellegi and Holt published their landmark paper on automatic edit and imputation in the Journal of the American Statistical Association (Fellegi and Holt 1976). That paper can be considered to be the starting point for modern systems for automatic data editing. It describes a paradigm for identifying errors in a record automatically. According to this paradigm the data of a record should be made to satisfy all edit rules (*edits* for short) by changing the values of the fewest possible number of variables. In due course the original Fellegi-Holt paradigm has been generalised to: the data of a record should be made to satisfy all edits by changing the values of the variables with the smallest possible sum of *reliability weights*. A reliability weight of a variable is a nonnegative number expressing how reliable one considers the values of this variable to be. A large reliability weight corresponds to a variable of which the values are considered trustworthy, and a small reliability weight to a variable of which the values are considered not so trustworthy. The (generalised) paradigm can be successfully applied to detect random, nonsystematic errors. A system based on the (generalised) Fellegi-Holt paradigm will in the remainder of this article be referred to as an FH-system.

Only a few statistical offices in the world have implemented an FH-system for

automatic edit and imputation. Examples of FH-systems for continuous data are GEIS (Kovar and Whitridge 1990) by Statistics Canada, SPEER (Winkler and Draper 1997) by the U.S. Bureau of the Census, AGGIES (Todaro 1999) by NASS, a SAS program developed by the Central Statistical Office of Ireland (Central Statistical Office 2000), and CherryPi (De Waal 1996) by Statistics Netherlands. Examples of FH-systems for categorical data are SCIA (Barcaroli et al. 1995) by ISTAT and DISCRETE (Winkler and Petkunas 1997) by the U.S. Bureau of the Census. It is quite surprising that so few statistical offices have developed FH-systems considering the potential benefits of such a system. Two reasons for the limited number of FH-systems can be pointed out. The first reason is that some statistical offices distrust the quality of automatically edited and imputed data. This is an important reason for not applying automatic editing and imputation. We feel, however, that if automatic editing is used in combination with selective editing and macro-editing, data of sufficiently high quality can be obtained.

The second reason for not developing and implementing a system for automatic editing and imputation is that this is considered too complicated by many statistical offices. In fact, even the offices that have developed a system for automatic editing and imputation have developed that system for either categorical data or continuous data. Developing an FH-system that can handle mixed data, i.e., a mix of categorical and continuous data, is generally considered too hard. To our knowledge only Sande (Sande 2000) has developed an FH-system for mixed data. That system is based on a vertex generation approach (Sande 1978; De Waal 2003a). GEIS, AGGIES, the SAS program developed by the Central Statistical Office of Ireland, and CherryPi are also based on Sande's ideas, but can handle only continuous data.

The hardest part of developing an FH-system consists of building a solver for the so-called error localisation problem, i.e., the problem of identifying the erroneous fields in an erroneous record. In their paper Fellegi and Holt already described a method for solving this problem. The method is based on the generation of so-called *implicit*, or *implied*, edits. These implicit edits are logically implied by the explicitly specified edits. After the so-called complete set of explicit and implicit edits has been determined it is fairly straightforward to solve the error localisation problem. For each faulty record one begins by determining the violated (explicit and implicit) edits. Now, any set of variables that covers the violated edits, i.e., any set of variables such that in each violated edit at least one variable from this set is involved, can be imputed consistently, i.e., such that all edits become satisfied. According to the (generalised) paradigm of Fellegi and Holt a set of variables with the minimum sum of reliability weights among the sets of variables that cover the violated edits should be selected for imputation. A drawback of the method of Fellegi and Holt is that there may be extremely many implied edits. In such a case the method may be impractical to use. In particular, generating the complete set of explicit and implicit edits for linear inequality situations for continuous data remains too computationally expensive for moderate to large size problems.

In the present article we consider the error localisation problem for mixed data. The aim of the article is to dispel the myth that a solver for the error localisation problem has to be very complicated. After we have given a mathematical description of the error localisation problem for mixed data in Section 2, we present an algorithm for solving this problem in Section 3. The proposed algorithm is rather simple, and for most statistical offices

implementing this algorithm should not be a major problem. The algorithm is based on a similar algorithm for purely numerical data (Quere 2000). Section 4 gives an example illustrating the algorithm. A proof that our algorithm finds all optimal solutions to the error localisation problem for mixed data is given in Section 5. Section 6 discusses performance issues related to the algorithm, and Section 7 contains a short discussion.

2. The Error Localisation Problem for Mixed Data

In this section we give a mathematical formulation of the error localisation problem for mixed data. We start by introducing some notation and terminology. We denote the categorical variables by v_i ($i = 1, \ldots, m$) and the continuous variables by x_k ($k = 1, \ldots, n$). For categorical data we denote the domain, i.e., the set of possible values, of variable i by D_i . The edits that we consider in this article are of the following type:

$$\text{IF } v_i \in F_i^j \text{ (for all } i = 1, \ldots, m)$$
$$\text{THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} \mid a_{1j}x_1 + \ldots + a_{nj}x_n + b_j \geq 0\} \tag{1a}$$

or

$$\text{IF } v_i \in F_i^j \text{ (for all } i = 1, \ldots, m)$$
$$\text{THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} \mid a_{1j}x_1 + \ldots + a_{nj}x_n + b_j = 0\} \tag{1b}$$

where $F_i^j \subset D_j$. All edits ($j = 1, \ldots, J$) given by (1) have to be satisfied simultaneously. We assume that the edits can indeed be satisfied simultaneously.

Examples of edits of Type (1) are:

a) IF ($v_i \in D_i$ for $i = 1, \ldots, m$) THEN *Turnover* = *Profit* + *Costs*
   This edit expresses that the turnover of an enterprise should equal the sum of profit and costs. It is in fact an example of a purely numerical edit.
b) IF ((*Gender* = *Male*) AND (*Pregnant* = *Yes*)) THEN $\varnothing$
   This edit expresses that males cannot be pregnant. It is an example of a purely categorical edit.
c) IF (*Activity* $\in$ {*Chemical Industry*, *Car Industry*}) THEN (*Turnover* $\geq$ 1,000,000 Euros)
   This edit expresses that an enterprise in the chemical industry or in the car industry should have a turnover of at least one million Euros.

A variable such as *Age* can either be considered to be a categorical variable or a continuous one, depending on the kind of edits involving *Age*. However, a variable cannot be considered to be a categorical variable in one edit and a continuous variable in another.

The condition after the IF-statement of (1), i.e., ''$v_i \in F_i^j$ for all $i = 1, \ldots, m$,'' is called the IF-condition of the edit. The condition after the THEN-statement is called the THEN-condition. If the IF-condition does not hold true, the edit is satisfied, irrespective of the values of the continuous variables. A categorical variable v_i is said to *enter* an edit j given by (1) if $F_i^j \subset D_i$ and $F_i^j \neq D_i$, i.e., if F_i^j is strictly contained in the domain of variable i. That edit is then said to *involve* this categorical variable. A continuous variable x_k is said to *enter* the THEN-condition of edit j given by (1) if $a_{kj} \neq 0$. That THEN-condition is

then said to *involve* this continuous variable. If F_i^j in the IF-condition of (1) is the empty set for some $i = 1, \ldots, m$ or the set in the THEN-condition is the entire n-dimensional real vector space, then the edit is always satisfied and may be discarded.

In many practical cases, certain kinds of missing values are acceptable, for instance when the corresponding questions are not applicable to a particular respondent. We assume that for categorical variables such acceptable missing values are coded by special values in their domains. Nonacceptable missing values of categorical variables are not coded. The optimisation problem formulated below will identify these missing values as being erroneous. We also assume that numerical THEN-conditions are only triggered if the value of none of the variables involved may be missing. Hence, if – for a certain record – a THEN-condition involving a numerical variable of which the value is missing is triggered by the categorical values, then either the missing numerical value is erroneous or at least one of the categorical values.

For each record $(v_1^0, \ldots, v_m^0, x_1^0, \ldots, x_n^0)$ in the data set that is to be edited automatically we have to determine a synthetic record $(v_1, \ldots, v_m, x_1, \ldots, x_n)$ such that (1) becomes satisfied for all edits $j = 1, \ldots, J$ and such that

$$\sum_{i=1}^{m} w_i^c \delta(v_i^0, v_i) + \sum_{k=1}^{n} w_k^r \delta(x_k^0, x_k)$$

is minimised. Here w_i^c is the nonnegative reliability weight of categorical variable i $(i = 1, \ldots, m)$, w_k^r the nonnegative reliability weight of numerical variable k $(k = 1, \ldots, n)$, $\delta(y^0, y) = 1$ if $y^0 \neq y$, and $\delta(y^0, y) = 0$ if $y^0 = y$. The variables of which the values in the synthetic record differ from the original values plus the variables for which the original values were missing together form an optimal solution to the error localisation problem. Note that the above formulation is a mathematical formulation of the generalised Fellegi-Holt paradigm. Note also that there may be several optimal solutions to a specific instance of the error localisation problem. Our aim is to find all these optimal solutions.

By generating all optimal solutions to the mathematical error localisation problem, we gain the option to later select one of these optimal solutions, using a secondary, more statistical criterion. The variables involved in the selected solution for which the values in the synthetic record differ from the original values are set to missing. The original values of these variables are considered to be incorrect and have to be imputed later by means of more appropriate values.

3. A Simple Algorithm for Automatic Editing of Mixed Data

In this section we give a mathematical description of our algorithm. The basic idea of the algorithm is that for each record a binary tree is constructed. Following Cormen, Leiserson, and Rivest (1990), we recursively define a binary tree as a structure on a finite set of nodes that either contains no nodes, or comprises three disjoint set of nodes: a root node, a left (binary) subtree and a right (binary) subtree. If the left subtree is nonempty, its root node is called the left child node of the root node of the entire tree, which is then called the parent node of the left child node. Similarly, if the right subtree is nonempty, its root node is called the right child node of the root node of the entire tree, which is

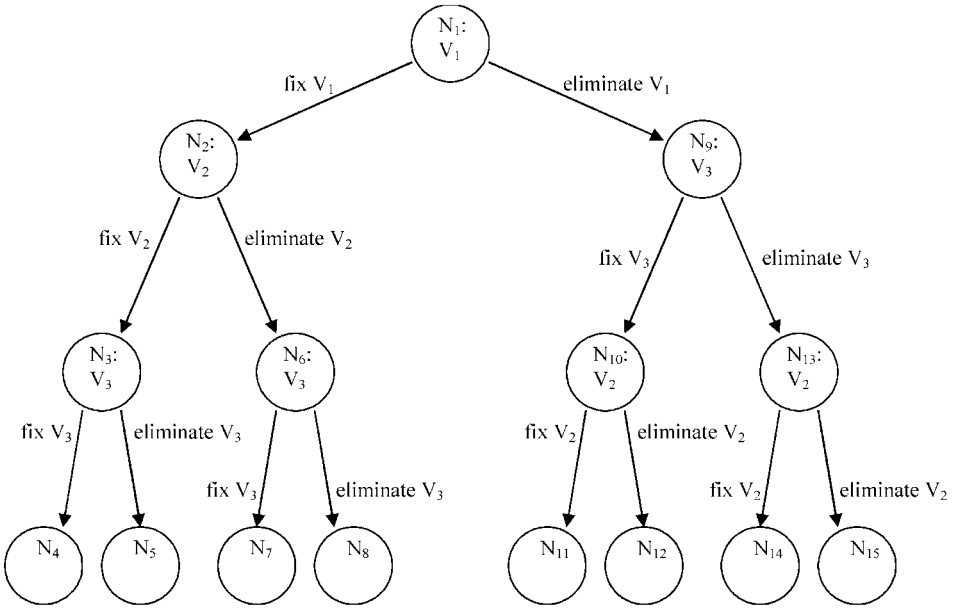

Fig. 1. A binary tree

then called the parent node of the right child node. All nodes except the root node in a binary tree have exactly one parent node. Each node in a binary tree can have at most two (nonempty) child nodes. A node in a binary tree that has only empty subtrees as its child nodes is called a terminal node, or also a leaf. A nonleaf node is called an internal node.

In Figure 1 we have drawn a small binary tree involving 15 nodes. Node N_1 is the root node of the entire binary tree. The child nodes of N_1 are N_2 and N_9. The terminal nodes are nodes N_4, N_5, N_7, N_8, N_{11}, N_{12}, N_{14}, and N_{15}.

In each node of the binary tree generated by our algorithm a variable is selected that has not yet been selected in any predecessor node. If all variables have already been selected in a predecessor node, we have reached a terminal node of the tree.

We first assume that no values are missing. After selection of a variable two branches are then constructed: in one branch the selected variable is fixed to its original value, in the other branch the selected variable is eliminated from the set of current edits. In each branch the current set of edits is updated. For instance, in node N_1 of Figure 1 variable V_1 is selected. In the left-hand branch variable V_1 is fixed to its original value, and in the right-hand branch V_1 is eliminated. A variable that has either been fixed or eliminated is said to have been treated (for the corresponding branch of the tree). The set of edits corresponding to the root node of our tree is the original set of edits. In this article we treat all continuous variables before any categorical variable is selected and treated. Fixing a variable to its original value corresponds to assuming that this original value is correct, and eliminating a variable from the set of current edits corresponds to assuming that the original value of this variable is incorrect and has to be modified.

Updating the set of current edits is the most important step in the algorithm. How the set of edits has to be updated depends on whether the selected variable is fixed or eliminated,

and also on whether this variable is categorical or continuous. Fixing a variable, either continuous or categorical, to its original value is easy. We simply substitute this value in all current edits, failing as well as nonfailing ones. Note that, conditional on fixing this variable to its original value, the new set of current edits is a set of implied edits for the remaining variables in the tree. That is, conditional on the fact that the selected variable has been fixed to its original value, the remaining variables have to satisfy the new set of edits. As a result of fixing the selected variable to its original value some edits may become satisfied, for instance when a categorical variable is fixed to a value such that the IF-condition of an edit can never become true any more. These edits may be discarded from the new set of edits. Conversely, some edits may become violated. In such a case this branch of the binary tree cannot result in a solution to the error localisation problem.

Eliminating a variable is a relatively complicated process. It amounts to generating a set of implied edits that do not involve this variable. In this generation process we need to consider both the failing edits and the nonfailing ones in the current set of edits. The generated set of implied edits becomes the set of edits corresponding to the new node of the tree. If a continuous variable is to be eliminated, we basically apply Fourier-Motzkin elimination (Duffin 1974) to eliminate that variable from the set of edits. Some care has to be taken in order to ensure that the IF-conditions of the resulting edits are correctly defined. In particular, if we want to eliminate a continuous variable x_r from the current set of edits, we start by copying all edits not involving this continuous variable from the current set of edits to the new set of edits. Next, we consider all edits in format (1) involving x_r pair-wise. Suppose we consider a pair of edits s and t. We start by checking whether the intersection of the IF-conditions is nonempty, i.e., whether the intersections $F_i^s \cap F_i^t$ are nonempty for all $i = 1, \ldots, m$. If any of these intersections is empty, we do not have to consider this pair of edits anymore. Now suppose that all intersections are nonempty. We then construct an implied edit. If the THEN-condition of edit s is an equality, we use the equality

$$x_r = -\frac{1}{a_{rs}} \left(b_s + \sum_{i \neq r} a_{is} x_i \right) \tag{2}$$

to eliminate x_r from the THEN-condition of edit t. Similarly, if the THEN-condition of edit s is an inequality and the THEN-condition of edit t is an equality, the equality in edit t is used to eliminate x_r. If the THEN-conditions of both edit s and edit t are inequalities, we check whether the coefficients of x_r in those inequalities have opposite signs. That is, we check whether $a_{rs} \times a_{rt} < 0$. If that is not the case, we do not consider this pair of edits any more. If the coefficients do have opposite signs, we can write one inequality as an upper bound on x_r and the other as a lower bound on x_r. We generate the following THEN-condition for our implied edit:

$$(x_1, \ldots, x_n) \in \{\mathbf{x} \mid \tilde{a}_1 x_1 + \ldots + \tilde{a}_n x_n + \tilde{b} \geq 0\}$$

where

$$\tilde{a}_i = |a_{rs}| \times a_{it} + |a_{rt}| \times a_{is} \qquad \text{for all } i = 1, \ldots, m$$

and

$$\tilde{b} = |a_{rs}| \times b_t + |a_{rt}| \times b_s$$

Note that x_r indeed does not enter the resulting THEN-condition. The IF-condition of the implied edit is given by the intersections $F_i^s \cap F_i^t$ for all $i = 1, \ldots, m$.

Note that if we eliminate a continuous variable in any of the ways described above, the resulting set of edits is a set of implied edits for the remaining variables in the tree. That is, this resulting set of edits has to be satisfied by the remaining variables in the tree.

In our algorithm, categorical variables are only treated, i.e., fixed or eliminated, once all continuous variables have been treated. This is done in order to keep our algorithm as simple as possible. If categorical variables were treated before all continuous ones have been treated, we could obtain edits that are more complex than the edits of Type (1) (see De Waal 2003b). So, once the categorical variables are selected, the edits in the current set of edits all have the following form:

$$\text{IF } v_i \in F_i^j \text{ (for } i = 1, \ldots, m) \text{ THEN } (x_1, \ldots, x_n) \in \varnothing \tag{3}$$

To eliminate categorical variable v_r from a set of edits given by (3), we start by copying all edits not involving this variable to the set of implied edits.

Next, we basically apply the method of Fellegi and Holt to the IF-conditions to generate the IF-conditions of the implied edits. In the terminology of Fellegi and Holt, field v_r is selected as the generated field. We start by determining all index sets S such that

$$\bigcup_{j \in S} F_r^j = D_i \tag{4}$$

and

$$\bigcap_{j \in S} F_i^j \neq \varnothing \qquad \text{for all } i = 1, \ldots, r-1, r+1, \ldots, m \tag{5}$$

From these index sets we select the *minimal* ones, i.e., the index sets S that obey (4) and (5), but where none of their subsets obey (4).

Given such a minimal index set S we construct the implied edit given by

$$\text{IF } v_r \in D_r, v_i \in \bigcap_{j \in S} F_i^j \text{ (for } i = 1, \ldots, r-1, r+1, \ldots, m)$$
$$\text{THEN } (x_1, \ldots, x_n) \in \varnothing \tag{6}$$

Note that if we eliminate a categorical variable in the way described above, the resulting set of edits is a set of implied edits for the remaining variables in the tree. That is, this resulting set of edits has to be satisfied by the remaining variables in the tree. We have now explained how the current set of edits changes if we fix or eliminate a variable.

If values are missing in the original record, the corresponding variables only have to be eliminated (and not fixed) from the set of edits, because these variables always have to be imputed. A natural choice is to treat the variables in the following order:

- eliminate all continuous variables with missing values;
- fix or eliminate the remaining continuous variables;
- eliminate all categorical variables with missing values;
- fix or eliminate the remaining categorical variables.

After all categorical variables have been treated we are left with a set of relations involving no unknowns. This set of relations may be the empty set, in which case it obviously does

not contain any self-contradicting relations. A self-contradicting relation is given by

IF $v_i \in D_i$ (for $i = 1, \ldots, m$) THEN $(x_1, \ldots, x_n) \in \varnothing$

The set of relations contains no self-contradicting relations if and only if the variables that have been eliminated in order to reach the corresponding terminal node of the tree can be imputed consistently, i.e., such that all original edits can be satisfied (cf. Theorems 1 and 2 in Section 5).

In the algorithm we check for each terminal node of the tree whether the variables that have been eliminated in order to reach this node can be imputed consistently. Of all the sets of variables that can be imputed consistently we select the ones with the lowest sum of reliability weights. In this way we find all optimal solutions to the error localisation problem (cf. Theorem 3 in Section 5).

Equalities in THEN-conditions can be handled more efficiently than we have described so far. For instance, if the numerical variable to be eliminated is involved in an equality that has to hold irrespective of the values of the categorical variables, i.e., is involved in an edit of the following type:

IF $v_i \in D_i$ (for $i = 1, \ldots, m$)

THEN $(x_1, \ldots, x_n) \in \{\mathbf{x} \mid a_{1s}x_1 + \ldots a_{ns}x_n + b_s = 0\}$ (7)

then we do not have to consider all edits pair-wise in order to eliminate this variable. Instead, we only have to combine (7) with all other current edits. So, if there are J current edits, we do not have to consider $J(J-1)$ pairs, but only $J-1$ pairs. Furthermore, the number of resulting implied edits is generally less than when all pairs of current edits are considered. We refer to this rule as the equality-elimination rule.

The algorithm sketched in this section is a so-called branch-and-bound algorithm. In a branch-and-bound algorithm a tree is constructed and bounds on the objective function are used to cut off branches of the tree. In Section 6 we explain how branches can be cut off from our tree.

4. Example

In this section we illustrate the algorithm presented in Section 3 by means of an example. We will not build the entire tree, because this would take too much space. Instead we will only generate one branch of the tree. Suppose we have to edit a data set containing two categorical variables v_i (i=1,2) and two numerical variables x_k ($k = 1, 2$). The domains of the two categorical variables are $D_1 = \{1, 2\}$ and $D_2 = \{1, 2, 3\}$, respectively. The set of explicit edits is the following.

IF ($v_1 = 1$ AND $v_2 \in D_2$) THEN $\varnothing$ (8)

IF ($v_1 \in D_1$ AND $v_2 \in D_2$) THEN $x_1 - 12 \geqslant 0$ (9)

IF ($v_1 \in D_1$ AND $v_2 \in \{1, 3\}$) THEN $x_2 = 0$ (10)

IF ($v_1 \in D_1$ AND $v_2 = 2$) THEN $x_2 - 1{,}250 \geqslant 0$ (11)

IF ($v_1 \in D_1$ AND $v_2 = 2$) THEN $-875x_1 + 12x_2 \geqslant 0$ (12)

IF ($v_1 \in D_1$ AND $v_2 = 2$) THEN $1{,}250x_1 - 8x_2 \geqslant 0$ (13)

Now, suppose that a record with values $v_1 = 1$, $v_2 = 2$, $x_1 = 25$, and $x_2 = 3{,}050$ is to be edited. Edit (8) is violated, so this record is inconsistent. We select a numerical variable, say x_1. Two branches are generated: one branch where x_1 is fixed to its original value, and one branch where x_1 is eliminated from the current set of edits. Here we only consider the branch where x_1 is eliminated. If we combine, for instance, edits (9) and (12), we first take the intersection of their IF-conditions. This intersection is given by ''$v_1 \in D_1$ AND $v_2 = 2$.'' This intersection is nonempty, so we proceed. The THEN-condition of the resulting implied edit is given by $12x_2 \geqslant 875 \times 12$, or equivalently by $x_2 \geqslant 875$. The resulting implied edit is hence given by

$$\text{IF } (v_1 \in D_1 \text{ AND } v_2 = 2) \text{ THEN } x_2 - 875 \geqslant 0 \tag{14}$$

The new set of (explicit and implicit) edits is given by (14),

$$\text{IF } (v_1 \in D_1 \text{ AND } v_2 = 2) \text{ THEN } x_2 \geqslant 0 \tag{15}$$

and (8), (10), and (11).

We select the other numerical variable x_2 and again construct two branches. Here we only consider the branch where x_2 is fixed to its original value. In this case, for instance edit (11) becomes satisfied and is discarded from the current branch of the tree. The resulting set of edits obtained by fixing x_2 to its original value is given by:

$$\text{IF } (v_1 \in D_1 \text{ AND } v_2 \in \{1, 3\}) \text{ THEN } \varnothing \tag{16}$$

and (8). Edit (16) arises from edit (10) by substituting 3,050 for x_2.

All numerical variables have now been treated, and we start treating the categorical variables. We select a categorical variable, say v_1, and again split the tree into two branches. Here, we only consider the branch where v_1 is eliminated. The resulting set of edits is given by edit (16) only. We select the other categorical variable, v_2, and again construct the two branches. Here, we only consider the branch where v_2 is fixed to its original value. The resulting set of edits is empty, which implies that the set of explicit, original edits can be satisfied by changing the values of x_1 and v_1 and fixing x_2 and v_2 to their original values. That is, a (possibly suboptimal) solution to the error localisation problem for this record is: change the values of x_1 and v_1. Possible values are $v_1 = 2$ and $x_1 = 40$. By examining all branches of the tree, including the ones we have skipped here, we can obtain all optimal solutions to the error localisation problem for the record under consideration.

5. An Optimality Proof

In this section we prove that the algorithm described in Section 3 indeed finds all optimal solutions to the error localisation problem. We do this in three steps.

1. Each time we eliminate or fix a variable the current set of edits is transformed into a new set of edits. The new set of edits involves at least one variable fewer than the current set of edits. We start by showing that the current set of edits can be satisfied if the new set of edits can be satisfied. This is the content of Theorem 1 below.
2. Using this result we show that if and only if the relations involving no unknowns in a terminal node do not contradict each other, we can impute the variables that have

been eliminated in order to reach this terminal node consistently, i.e., such that the original edits become satisfied. This is the content of Theorem 2.

3. The final step consists of observing that the terminal nodes correspond to all potential solutions to the error localisation problem, and hence that the algorithm determines all optimal solutions to the error localisation problem. This is the content of Theorem 3.

Steps 2 and 3 are trivial once the first step has been proved. The proof of the first step is similar to the proof of Theorem 1 in Fellegi and Holt (1976). The main differences are that the edits considered by Fellegi and Holt differ from the edits considered in the present article, and that Fellegi and Holt assume that the so-called complete set of (explicit and implied) edits has been generated. We will not make this assumption.

THEOREM 1. Suppose the index set of the variables in a certain node is given by T_0, and the current set of edits corresponding to that node by Ω_0. Suppose furthermore that to obtain a next node a certain variable r is either fixed or eliminated. Denote the index set of the resulting variables by T_1 ($T_1 = T_0 - \{r\}$) and the set of edits corresponding to this next node by Ω_1. Now, if there exist values u_i for $i \in T_1$ that satisfy the edits in Ω_1, then there exists a value u_r for variable r such that the values u_i for $i \in T_0$ satisfy the edits in Ω_0.

Proof. We distinguish between several cases. First, let us suppose that the selected variable is fixed. This is a trivial case. It is clear that if there exist values u_i for $i \in T_1$ that satisfy the edits in Ω_1, there exist values u_i for $i \in T_0$ that satisfy the edits in Ω_0. Namely, for the fixed variable r we set the value u_r equal to the original value of r.

Let us now suppose that a categorical variable r has been eliminated. Note that in our algorithm all continuous variables have then already been either fixed or eliminated. Suppose that there exist values u_i for $i \in T_1$ that satisfy the edits in Ω_1, but there does not exist a value u_r for the selected variable r such that the values for $i \in T_0$ satisfy the edits in Ω_0. Set the values for $i \in T_1$ to u_i, and identify a failed edit in Ω_0 for each possible value of variable r. The index set of these failed edits need not be a minimal one. We therefore remove some of the failed edits such that the corresponding index set S becomes minimal. We then construct the implied edit given by (6). Edit (6) is an element of Ω_1. Moreover, the values u_i for $i \in T_1$ do not satisfy this edit. This contradicts our assumption that these values satisfy all edits in Ω_1. So we can conclude that a value u_r for the selected variable r exists such that the values u_i for $i \in T_0$ satisfy the edits in Ω_0.

Now, let us suppose that a continuous variable r has been eliminated. Suppose that there exist values u_i for $i \in T_1$ that satisfy the edits in Ω_1. Each edit in Ω_1 is obtained either from copying the edits in Ω_0 not involving variable r, or from two edits in Ω_0 involving variable r that have been combined.

It is clear that if the edits in Ω_1 that have been obtained from copying the edits in Ω_0 not involving variable r are satisfied by the values u_i for $i \in T_1$, these edits in Ω_0 are also satisfied by the same values for $i \in T_0$.

It remains to be proved that if the edits in Ω_1 that have been obtained by combining two edits in Ω_0 are satisfied by $i \in T_1$, there exists a value u_r for variable r such that all edits in

Ω_0 involving variable r can be satisfied. To prove this statement, we substitute the values u_i for $i \in T_1$ into the edits in Ω_0. As a result, we obtain a number of constraints for the value of the selected variable r. Such a constraint can be an equality involving x_r, a lower bound on x_r, or an upper bound on x_r. That is, these constraints are given by:

$$x_r = M_j^E \tag{17}$$

$$x_r \geqslant M_j^L \tag{18}$$

and

$$x_r \leqslant M_j^U \tag{19}$$

where M_j^E, M_j^L and M_j^U are certain constants, and j is an index for the edits in Ω_0 involving variable r.

A constraint of type (17) has been obtained from an edit j in Ω_0 of which the THEN-condition can be written in the following form

$$x_r = \sum_{i \neq r} a'_{ij} x_i + b'_j \tag{20}$$

by filling in the values u_i for $i \in T_1$. Similarly, constraints of types (18) and (19) have been obtained from edits in Ω_0 of which the THEN-conditions can be written as $x_r \geqslant \sum_{i \neq r} a'_{ij} x_i + b'_j$ and $x_r \leqslant \sum_{i \neq r} a'_{ij} x_i + b'_j$, respectively, by filling in the values u_i for $i \in T_1$.

If the constraints given by (17) to (19) do not contradict each other, we can find a value u_r for variable r such that this value plus the values u_i for $i \in T_1$ satisfy the edits in Ω_0. Namely, select any value u_r for variable r such that all constraints (17) to (19) are satisfied. Suppose the constraints given by (17) to (19) contradict each other. These constraints can only contradict each other if there are constraints s and t given by

1. $x_r = M_s^E$ and $x_r = M_t^E$ with $M_s^E \neq M_t^E$
2. $x_r = M_s^E$ and $x_r \geqslant M_t^L$ with $M_s^E < M_t^L$,
3. $x_r \leqslant M_s^U$ and $x_r = M_t^E$ with $M_s^U < M_t^E$

or

4. $x_r \leqslant M_s^U$ and $x_r \geqslant M_t^L$ with $M_s^U < M_t^L$

In Case 1 constraints s and t have been derived from edits in Ω_0 of which the THEN-conditions are equalities. The IF-conditions of these edits have a nonempty intersection, because both edits are triggered when we fill in the values u_i for the categorical variables in T_1. So, these edits generate an implied edit in Ω_1 if we eliminate variable r. The THEN-condition of this implied edit can be written as

$$\sum_{i \neq r} a'_{is} x_i + b'_s = \sum_{i \neq r} a'_{it} x_i + b'_t$$

where we have used (20).

Filling in the values u_i for $i \in T_1$ in this implied edit, we find that M_s^E should be equal to M_t^E. In other words, we have constructed an edit in Ω_1 that would fail if we filled in the values u_i for $i \in T_1$. This contradicts our assumption that these values satisfy all edits

in Ω_1, and we conclude that two constraints given by (17) (Case 1 above) cannot contradict each other.

For Cases 2, 3, and 4 we can show in a similar manner that we would be able to construct a failed implied edit in Ω_1. This contradicts our assumption that the values u_i for $i \in T_1$ satisfy all edits in Ω_1, and we conclude that the constraints given by (17) to (19) cannot contradict each other. In turn this allows us to conclude that a value u_r for variable r exists such that this value plus the values u_i for $i \in T_1$ satisfy the edits in Ω_0.

Finally, note that if an edit (7) has been used to eliminate a continuous variable r by means of the equality-elimination rule, the value given by

$$x_r = -\frac{1}{a_{rs}} \left(b_s + \sum_{i \neq r} a_{is} u_i \right)$$

plus the values u_i for $i \in T_1$ satisfy the edits in Ω_0.

In the Appendix below, we give the last part of the proof of Theorem 1.

This concludes the proof of Theorem 1.

THEOREM 2. If and only if the set of edits corresponding to a terminal node – a set of relations involving no unknowns – is consistent, then the variables that have been eliminated in order to reach this terminal node can be imputed in such a way that the original set of edits is satisfied.

Proof. This follows directly from a repeated application of Theorem 1, and the fact that eliminating a variable amounts to generating a set of implied edits for the remaining variables.

THEOREM 3. The algorithm described in Section 3 determines all optimal solutions to the error localisation problem.

Proof. The terminal nodes of the tree correspond to all possible combinations of fixing and eliminating variables. Thus, according to Theorem 2 above, the algorithm checks which of all possible sets of variables can be imputed consistently. The algorithm simply selects all optimal sets of variables that can be imputed consistently from all possible sets of variables. Therefore we can conclude that the algorithm finds all optimal solutions to the error localisation problem.

6. Computational Results

We have demonstrated in Section 5 that our algorithm determines all optimal solutions to the error localisation problem for mixed data. At first sight, however, the developed algorithm may seem rather slow because an extremely large binary tree has to be generated to find all optimal solutions, even for moderately sized problems. Fortunately, the situation is not nearly as bad as it may seem. First of all, if the minimum number of fields that has to be changed in order to make a record pass all edits is (too) large, the record should not be edited automatically in our opinion. We consider the quality of such a record to be too low to correct it automatically. In our opinion, such a record should either be edited manually, or be discarded completely. By specifying an upper bound on the number of fields that may be changed, the size of the tree can drastically be reduced.

Table 1. Characteristics of the data sets

	Data set A	Data set B	Data set C	Data set D	Data set E	Data set F
Number of variables	90	76	53	51	54	26
Number of nonnegativity constraints	90	70	36	49	54	22
Number of balance edits[a]	0	18	20	8	21	3
Number of inequality edits[b]	8	2	16	7	0	15
Total number of records	4,347	274	1,480	4,217	1,039	1,425
Number of inconsistent records[c]	4,347	157	1,404	2,152	378	1,141
Total number of missing values	259,838	0	0	0	2,230	195
Number of records with more than 6 errors or missing values	4,346	7	117	16	136	8
Errors per inconsistent record[d]	0.2	2.5	2.6	1.6	5.8	3.0
Number of optimal solutions per inconsistent record	6.1	12.0	6.9	23.3	1.2	11.6

[a] A balance edit is an edit of Type (1b).
[b] Excluding nonnegativity constraints.
[c] Including records with missing values.
[d] Excluding missing values.

The size of the tree can also be reduced during the execution of the algorithm, because it may already become clear in an internal node of the tree that the corresponding terminal nodes cannot generate an optimal solution to the problem. For instance, by fixing the wrong variables we may make the set of edits infeasible.

The value of the objective function can also be used to reduce the size of the tree. This value cannot decrease while going down the tree. So, if the value of the objective function in an internal node exceeds the value of an already found (possibly suboptimal) solution, we can again conclude that the corresponding terminal nodes cannot generate an optimal solution to the problem. These terminal nodes need not be examined, and can be cut off the tree.

Because the size of the tree, and hence the computing time of the algorithm, can be influenced by the order in which the variables are treated, this ordering is very important in practice. The ordering must not be fixed before the execution of the algorithm as this would lead to an extensive average computing time. Instead the ordering should be determined dynamically, i.e., during the execution of the algorithm. Each time a variable is to be treated the ''best'' variable, according to a suitable ordering strategy, should be selected.

The algorithm described in this article has been implemented in a prototype program called Leo. This program has been compared to two other programs. For Statistics Netherlands improving the efficiency of the data editing process for economic, and hence mainly numerical, data is much more important than improving it for social, and hence mainly categorical, data. Therefore the developed programs have only been evaluated for purely numerical test data. For our evaluation experiments we have used six realistic data sets. In Table 1 we give a summary of the characteristics of the six data sets.

The numbers in the last two rows of Table 1 have been determined by comparing the number of fields involved in the optimal solutions, respectively the number of optimal solutions, of the three algorithms. The number of fields involved in the optimal solutions is assumed to be equal to the actual number of errors.

The six data sets come from a wide range of business surveys, such as a survey on labour costs, a structural business survey on enterprises in the photographic sector, a structural business survey on enterprises in the building and construction industry, and a structural business survey on the retail sector. Besides these data sets we have used two business data sets, the Swiss Environmental Protection Expenditure survey (data set E) and a subset of the UK Annual Business Inquiry (data set F), from the EUREDIT project, a research project under the Information Society Technologies Programme of the Framework Programme 5 of the European Union. The numbers of variables, edits and records are in most of the six data sets quite realistic for business surveys at Statistics Netherlands. Exceptions are data set A, where the number of edits other than nonnegativity edits is very small and the number of missing values is very large (almost 60 missing values per record on average, and only one record has fewer than 6 missing values), and data set B, where the number of records is very small. At Statistics Netherlands a very large and complex data set to be edited automatically may involve slightly more than 100 variables, about 100 edits, and a few thousand records.

For a detailed description of the edits of the six data sets, we refer to De Waal (2003b). For some data sets, for instance data sets C and E, all variables are interconnected by edits. Some of the other sets of edits, for instance those of data sets A and D, can, in principle, be split up into subsets involving disjoint sets of variables. This has not been done, however, which means that the applied algorithms (represent an attempt to) solve the entire error localisation problem at once rather than solving it for each subset of edits separately.

The first algorithm we consider is based on a standard mixed integer programming (MIP) formulation for the error localisation problem for continuous data (cf. De Waal 2003b). This algorithm has been implemented in Visual C++ 6.0, and calls routines of CPlex, a well-known commercial MIP-solver, to actually solve the MIP problems involved. We refer to this program as ERR_CPLEX in the remainder of this article. ERR_CPLEX finds only one optimal solution to each instance of the error localisation problem. To find all optimal solutions we could – once an optimal solution to the current MIP problem has been determined – iteratively add an additional constraint that basically states that the present optimal solution is excluded but other optimal solutions to the current MIP problem remain feasible, and solve the new MIP problem. This process of determining an optimal solution to the current MIP problem and adding an additional constraint to obtain a new MIP problem goes on until all optimal solutions to the error localisation problem have been found. We have not implemented this option, however. Resolving the problem from scratch for each optimal solution would be very time-consuming. The alternative is to use a so-called hot restart, where information generated to obtain an optimal solution to a MIP problem is utilised to obtain an optimal solution to a slightly modified MIP problem. A problem with this possibility is that experiences at Statistics Netherlands with CPlex so far, on linear programming problems arising in statistical disclosure control, shows that CPlex becomes numerically unstable if too many hot restarts in a row are applied. Our results for ERR_CPLEX are therefore only indicative.

The second algorithm is based on vertex generation (cf., Sande 1978; De Waal 2003a). This algorithm has been implemented in a program, CherryPi, using Delphi 3. The implemented algorithm uses a matrix to solve the error localisation problem. The number of rows of this matrix is implied by the number of edits and the number of variables. The

actual number of columns is determined dynamically. Owing to memory and speed restrictions a maximum for the allowed number of columns is set in CherryPi. If the actual number of columns exceeds the allowed maximum, certain columns are deleted. This influences the solutions that are found by CherryPi. Owing to this pragmatic rule in some cases only nonoptimal solutions may be found, and in some other cases no solutions at all may be found. Another effect of this pragmatic rule is that if columns have been deleted in order to find solutions to an instance of the error localisation problem, the optimality of the solutions found is not guaranteed. The larger the allowed number of columns, the better the quality of the solutions found by CherryPi, but also the slower the speed of the program. Practical experience has taught us that in many instances setting the allowed number of columns to 4,000 gives an acceptable trade-off between the quality of the solutions found and the computing time of the program. In the version of CherryPi that was used for the comparison study the allowed number of columns was therefore set to 4,000.

The third algorithm is based on the branch-and-bound approach proposed in this article. The algorithm has been implemented in the program Leo, using Delphi 3. The program requires that a maximum cardinality N_{max} for the optimal solutions should be specified beforehand. Only if a record can be corrected by N_{max} or fewer changes, are optimal solutions to the error localisation problem determined. For Leo we have performed two kinds of experiments: for the first kind N_{max} was set to 6, and for the second kind N_{max} was set higher. In Leo the following rule to select a branching variable has been implemented: first select the variables that are involved in at least one failed edit and a minimum number of satisfied edits, then select the variable from this set of variables that occurs most often in the failed edits. If there are several ''best'' variables to branch on, one of them is chosen randomly. In Leo, the equality-elimination rule described at the end of Section 3 has not been implemented. On two data sets, the data sets for which the computing times of Leo are comparatively bad, we have applied a special, alternative version of Leo in which the equality-elimination rule has been implemented. Leo sometimes suffers from memory problems, especially for records with many errors, because too many nodes with too many edits need to be stored. For records for which Leo suffers from memory problems, it cannot determine an optimal solution.

ERR_CPLEX, CherryPi, and Leo suffer from some numerical problems. These problems arise because in (erroneous) records the largest values may be a factor 10^9 or larger

Table 2. Computational results of the error localisation algorithms for the data sets (in seconds)

	Data set A	Data set B	Data set C	Data set D	Data set E	Data set F
ERR_CPLEX[1]	233 (1)	10 (0)	93 (1)	108 (8)	13 (0)	35 (0)
CherryPi	570 (38)	96 (1)	540 (7)	498 (30)	622 (3)	79 (0)
Leo[2]	18 (0)	308 (10)	531 (4)	21 (1)	59 (34)	7 (0)
Leo ($N_{max} = 6$)	7 (0)	51 (1)	94 (2)	19 (0)	4 (1)	8 (1)

[1] Tests performed on a special server. On this PC the only fully licensed version of CPlex at Statistics Netherlands has been installed. To compare the computing times of ERR_CPLEX to those of the other programs, they have been multiplied by a factor of approximately 1.08.

[2] To find the results for Leo for $N_{max} > 6$, we have set N_{max} equal to 90 for data set A, to 8 for data sets B and C, and to 12 for data sets D, E, and F.

than the smallest values. For instance, because of these numerical problems ERR_CPLEX occasionally generates suboptimal solutions containing too many variables.

The experiments have been performed on a 1,500 MHz PC with 256 MB of RAM. This PC is connected to a local area network. Computing times may therefore be influenced by the amount of data that was transmitted through the network at the time of the experiments. To reduce and to estimate this influence we have performed five experiments per data set at various times during the day. In Table 2 we present the average computing times of these experiments for the six data sets, and between brackets the standard deviations of these computing times over the corresponding five experiments are provided. In all experiments, the reliability weight of each variable was set to 1.

The values of $N_{\max}$ for Leo have been determined by trial and error. We have set $N_{\max}$ as high as possible without encountering memory problems for many, i.e., 20 or more, records. The value of $N_{\max}$ for data set A was set so high because the number of missing values for this data set are extremely large. Data set A was the most difficult one for ERR_CPLEX. It could not find any optimal solution for three records. Data set C was the most difficult one for CherryPi and Leo: CherryPi did not find a solution for 11 records, and Leo_8, i.e., Leo with $N_{\max} = 8$, not for 58 records. For 9 of these 58 records Leo_8 suffered from memory problems. Those 9 records were excluded from the computational results for Leo. Data set E was also difficult for Leo. Leo with $N_{\max} = 12$ suffered from memory problems for 13 records. For the other data sets Leo did not suffer from memory problems. Leo_6, i.e., Leo with $N_{\max} = 6$, did not suffer from memory problems for any of the six data sets.

In the context of special transportation problems and pure fixed charge transportation problems – problems that are similar to the error localisation problem – McKeown (1981) remarks that ''It is unclear in any of these contexts as to what makes a problem ''easy'' or ''difficult'' to solve.'' This remark has been confirmed for the error localisation problem. From the characteristics of the data sets it is hard to establish beforehand whether the corresponding instances of the error localisation problem will be ''easy'' or ''hard.'' For instance, in examining the characteristics of data sets B, C, and D, one might expect to see a similar performance per record, but this is not the case.

As already mentioned, the equality-elimination rule described at the end of Section 3 has not been implemented in Leo. For the two data sets for which ERR_CPLEX is better than Leo_6, data sets B and C, we have applied a special version of Leo in which this rule has been implemented. The results are given in Table 3. In this table we present the average computing times for the two data sets over five experiments, and between brackets the standard deviations of these computing times are provided.

For data set B the special version of Leo_8 could not find a solution for one record. For data set C the special version of Leo_8 could not find optimal solutions for 58 records, just

Table 3. Computational results for Leo with equality-elimination rule (in seconds)

	Data set B	Data set C
Leo with equality-elimination[1]	14 (2)	77 (1)
Leo with equality-elimination ($N_{\max} = 6$)	4 (1)	19 (1)

[1] To find the results for Leo for $N_{\max} > 6$, we have set $N_{\max}$ equal to 8 for both data sets.

like the standard version of Leo. The version of Leo with the equality-elimination rule did not suffer from memory problems, however.

Examining the results in Tables 2 and 3, we can conclude that as far as computing speed is concerned ERR_CPLEX and Leo (either with $N_{max} = 6$ or with $N_{max} > 6$) are the best programs. We note at the same time, however, that this conclusion is not completely justified as ERR_CPLEX determines only one optimal solution whereas the other programs (are designed to) determine all optimal solutions. Comparing the results of Tables 2 and 3, we see that the equality-elimination rule described at the end of Section 3 leads to a substantial reduction in computing time, at least for the two data sets examined. With this rule, Leo_6 is clearly faster than ERR_CPLEX for all data sets.

Besides computing speed other aspects are, of course, important too. We note that all programs, even the commercially available CPlex, suffer from numerical problems. As already mentioned, Leo in addition suffers from some memory problems. Because of its matrix with a fixed maximum number of columns, CherryPi does not always determine optimal solutions, but less good, suboptimal solutions. Summarising, it is hard to give a verdict on the quality of the solutions found by the programs as the programs suffer from a diversity of problems.

7. Discussion

In this article we have compared computing times of our branch-and-bound algorithm to those of other algorithms for solving the error localisation problem. It should be borne in mind, however, that these results do not say anything about the quality of the solutions in terms of their accuracy and their effects on the data. To evaluate these statistical aspects of automatic editing, several evaluation studies have been carried out at Statistics Netherlands and others are in progress. Unfortunately, most of our evaluation reports on automatic editing are in Dutch. Two exceptions are Houbiers, Quere, and De Waal (1999), and Hoogland and Van der Pijll (2003).

We consider the branch-and-bound algorithm described in the present article a very promising one for solving the error localisation problem. The main reason for our choice is the excellent performance of Leo for records with up to 6 errors. For such records it determines all optimal solutions very fast. We admit that for records with more than six errors the results of Leo become less good, just like the other algorithms. The program begins to suffer from memory problems, and the computing time increases. To some extent these problems can be overcome by implementing the equality-elimination rule described at the end of Section 3, and by storing edits in a more compact manner than we do in our prototype version. Besides, as we argued before, we feel that records with many errors should not be edited in an automatic manner, but manually. Given this point of view, Leo seems to be an excellent choice.

The proposed branch-and-bound algorithm is not very complex to implement and maintain. One of the reasons for the simplicity of the algorithm is that it is a very ''natural'' one. For instance, in the algorithm categorical and continuous variables are treated in almost the same manner, only the underlying method for generating implicit edits differs. Moreover, searching for optimal solutions to the error localisation problem is also a natural process. All possible solutions are simply checked, and the best solutions found are the

optimal ones. Because of the simplicity of the branch-and-bound algorithm discussed in this article, maintaining software based on it is relatively simple. Not only operations research specialists can understand the algorithm in detail, but also the IT specialists who develop and maintain the final computer program based on the mathematical algorithm.

Statistics Netherlands has therefore decided to implement this algorithm in a module of version 1.5 of our SLICE system (cf., De Waal 2001). This version reads an upper bound on the number of missing values per record as well as a separate upper bound on the number of errors (excluding missing values) per record. The former number is allowed to be quite large, say 50 or more, whereas the latter number is allowed to be moderate, say 10. If the number of missing values or the number of errors (excluding missing values) in a record exceeds either of these upper bounds, this record is rejected for automatic editing. Records that require too much computer memory are also rejected for automatic editing. The module does not have to be rerun when such records are detected, but simply skips the records once they are detected. The new module includes the equality-elimination rule. In addition, it contains a heuristic to handle integer data. Hence the new module solves the error localisation problem for a mix of categorical, continuous and integer data.

One may argue that some users of SLICE will want to edit records with many erroneous fields automatically despite our arguments against editing such records. These users might then be disappointed, because the new module will not be able to handle such records. To overcome this problem, we propose using a simple heuristic treatment of these records instead of applying the new module. For purely numerical data one could, for instance, minimise the sum of the absolute differences between the original values and the final values subject to the condition that all edits are satisfied. The resulting mathematical problem can be formulated as a linear programming problem, and can be solved quickly. For details on this linear programming approach and some suggestions for other heuristic approaches we refer to De Waal (2003b).

We are willing to admit that our choice of branch-and-bound algorithm is to some extent a subjective one, but we feel that it is a justifiable one.

8. References

Barcaroli, G., Ceccarelli, C., Luzi, O., Manzari, A., Riccini, E., and Silvestri, F. (1995). The Methodology of Editing and Imputation of Qualitative Variables Implemented in SCIA. Internal Report, ISTAT.

Central Statistical Office (2000). Editing and Calibration in Survey Processing. Report SMD-37, Ireland.

Cormen, T.H., Leiserson, C.E., and Rivest, R.L. (1990). Introduction to Algorithms. The MIT Press/McGraw-Hill Book Company.

De Waal, T. (1996). CherryPi: A Computer Program for Automatic Edit and Imputation. UN/ECE Work Session on Statistical Data Editing, Voorburg.

De Waal, T. (2001). SLICE: Generalised Software for Statistical Data Editing. Proceedings in Computational Statistics (eds. J.G. Bethlehem and P.G.M. Van der Heijden). Physica-Verlag, New York, 277–282.

De Waal, T. (2003a). Solving the Error Localization Problem by Means of Vertex Generation. Survey Methodology, 29, 71–79.

De Waal, T. (2003b). Processing of Erroneous and Unsafe Data. Ph.D. Thesis, Erasmus University, Rotterdam.

Duffin, R.J. (1974). On Fourier's Analysis of Linear Inequality Systems. Mathematical Programming Studies, 1, 71–95.

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

Granquist, L. (1990). A Review of Some Macro-Editing Methods for Rationalizing the Editing Process. Proceedings of the Statistics Canada Symposium, 225–234.

Granquist, L. (1995). Improving the Traditional Editing Process. In Business Survey Methods (eds. Cox, Binder, Chinnappa, Christianson and Kott), John Wiley and Sons, Inc., 385–401.

Granquist, L. (1997). The New View on Editing. International Statistical Review, 65, 381–387.

Granquist, L. and Kovar, J. (1997). Editing of Survey Data: How Much Is Enough? In Survey Measurement and Process Quality (eds. Lyberg, Biemer, Collins, De Leeuw, Dippo, Schwarz, and Trewin), John Wiley and Sons, Inc., 415–435.

Hedlin, D. (2003). Score Functions to Reduce Business Survey Editing at the U.K. Office for National Statistics. Journal of Official Statistics, 19, 177–199.

Hoogland, J. and Van der Pijll, E. (2003). Summary of the Evaluation of Automatic Versus Manual Editing of the Production Statistics 2000 Trade and Transport. UN/ECE Work Session on Statistical Data Editing, Madrid.

Houbiers, M., Quere, R., and De Waal, T. (1999). Automatically Editing the 1997 Survey on Environmental Costs. Internal report (BPA number: 4917-99-RSM), Statistics Netherlands, Voorburg.

Kovar, J. and Whitridge, P. (1990). Generalized Edit and Imputation System. Overview and Applications. Revista Brasileira de Estadistica, 51, 85–100.

Lawrence, D. and McKenzie, R. (2000). The General Application of Significance Editing. Journal of Official Statistics, 16, 243–253.

McKeown, P.G. (1981). A Branch-and-Bound Algorithm for Solving Fixed Charge Problems. Naval Research Logistics Quarterly, 28, 607–617.

Quere, R. (2000). Automatic Editing of Numerical Data. Report (research paper 0016), Statistics Netherlands, Voorburg.

Sande, G. (1978). An Algorithm for the Fields to Impute Problems of Numerical and Coded Data. Report, Statistics Canada.

Sande, G. (2000). Personal Communication.

Todaro, T.A. (1999). Overview and Evaluation of the AGGIES Automated Edit and Imputation System. UN/ECE Work Session on Statistical Data Editing, Rome.

Winkler, W.E. and Draper, L.A. (1997). The SPEER Edit System. Statistical Data Editing (Volume 2); Methods and Techniques. United Nations.

Winkler, W.E. and Petkunas, T.F. (1997). The DISCRETE Edit System. Statistical Data Editing (Volume 2); Methods and Techniques. United Nations.

Received December 2001
Revised July 2003