SORT 29 (1) January-June 2005, 57-100

Automatic error localisation for categorical, continuous and integer data

Ton de Waal*

Netherlands

Abstract

Data collected by statistical offices generally contain errors, which have to be corrected before reliable data can be published. This correction process is referred to as statistical data editing. At statistical offices, certain rules, so-called edits, are often used during the editing process to determine whether a record is consistent or not. Inconsistent records are considered to contain errors, while consistent records are considered error-free. In this article we focus on automatic error localisation based on the Fellegi-Holt paradigm, which says that the data should be made to satisfy all edits by changing the fewest possible number of fields. Adoption of this paradigm leads to a mathematical optimisation problem. We propose an algorithm for solving this optimisation problem for a mix of categorical, continuous and integer-valued data. We also propose a heuristic procedure based on the exact algorithm. For five realistic data sets involving only integer-valued variables we evaluate the performance of this heuristic procedure.

MSC: 03B05, 03B35, 62-02, 68T15, 90C10, 90C11

Keywords: branch-and-bound, categorical data, continuous data, error localisation, Fourier-Motzkin elimination, integer-valued data, statistical data editing.

1 Introduction

Data collected by statistical offices generally contain errors. In order to be able to publish reliable statistical information these errors have to be corrected. This correction process is referred to as statistical data editing. At statistical offices, certain rules, so-called edits, are often used to determine whether a record, *i.e*. the data of an individual respondent, is consistent or not. An example of such an edit is that the sum of the profit and the

* *Address for correspondence:* Statistics Netherlands. PO Box 4000. 2270 JM Voorburg. Netherlands, tel. +31 70 337 4930, fax +31 70 337 5964, e-mail: twal@cbs.nl.

Received: August 2004

Accepted: January 2005

costs of an enterprise should equal its turnover. Inconsistent records are considered to contain errors, while consistent records are considered error-free. If a record contains errors, the erroneous fields in this record need to be identified. In former days and often still nowadays, detected errors or inconsistencies were reported and explained on paper computer output or on a computer screen. Subsequently, subject-matter specialists corrected the errors by consulting the questionnaire, or by re-contacting the supplier of the information. This traditional form of statistical data editing, called manual (or interactive) editing, leads to statistical data of good quality, but is very costly in terms of resources and timeliness.

Several studies (*cf.* Granquist, 1995, 1997; Granquist and Kovar, 1997) have demonstrated that in order to obtain reliable publication figures only the most influential errors have to be edited manually. This observation, which has been confirmed by practical experience at several statistical offices, allows one to improve the efficiency of the statistical data editing process. For instance, at Statistics Netherlands most structural business surveys are nowadays treated by a combination of selective (or significance) editing (*cf.* Lawrence and McKenzie, 2000; Hoogland, 2002; Hedlin, 2003), automatic editing, and macro-editing (*cf.* Granquist, 1990). After data entry, simple checks, such as range checks, and simple automatic corrections, for instance in cases where a respondent filled in a financial figure in Euros instead of the requested thousands of Euros, are applied. Next, selective editing is applied to split the data into a critical stream and a non-critical stream. The former stream consists of those records that are the most likely ones to contain influential errors; the latter stream consists of the remaining records. The records in the critical stream are edited in a traditional, manual manner. The records in the non-critical stream are edited automatically. The final editing step we apply is macro-editing. Macro-editing consists of verifying whether the figures to be published seem plausible. Macro-editing is applied after outliers have been detected, raising weights have been determined, and the publication figures have been computed. It can lead to the detection of errors that would go unnoticed with selective editing or automatic editing. Only after the macro-editing step has been successfully completed can the publication figures be published. For more information on the statistical data editing strategy of Statistics Netherlands for structural business surveys we refer to De Jong (2002) and Hoogland (2002).

In this article we focus on automatic editing. Generally speaking, statistical data editing can be subdivided into the error localisation step and the imputation step. In the error localisation step the errors in the data are detected, in the imputation step the erroneous data are replaced by more accurate data and the missing values are filled in. To automate the statistical data editing process, both error localisation and imputation need to be automated.

In this article we restrict ourselves to discussing how to automate the error localisation step. To this end, one generally uses the (generalised) paradigm of Fellegi and Holt (1976) as a guiding principle to identify the errors. This (generalised) paradigm

says that the data of a record should be made to satisfy all edits by changing the fewest possible (weighted) number of fields. Here each variable is given a weight, the so-called reliability weight, which is a measure for the level of confidence in the values of this variable. The higher the reliability weight of a variable, the more reliable its observed values are considered to be. In the original form of the Fellegi-Holt paradigm each variable was given a reliability weight of 1. Using the (generalised) Fellegi-Holt paradigm, as we do in the present article, the error localisation problem can be formulated as a mathematical optimisation problem.

Solving this mathematical optimisation problem is, however, a non-trivial matter. Overviews of various algorithms for solving the error localisation problem based on the Fellegi-Holt paradigm have been given by Liepins, Garfinkel and Kunnathur (1982), and De Waal and Coutinho (2005). Most algorithms and software packages for solving this problem described in the literature are either designed for categorical (discrete) data or for continuous data. Algorithms for categorical data have been proposed by Fellegi and Holt (1976), Garfinkel, Kunnathur and Liepins (1986), Winkler (1998), Bruni, Reale and Torelli (2001), Bruni and Sassano (2001), and Boskovitz, Goré and Hegland (2003). Software packages for categorical data include SCIA (*cf.* Barcaroli *et al.*, 1995) by ISTAT, and DISCRETE (*cf.* Winkler and Petkunas, 1997) by the US Bureau of the Census. Algorithms for solving the problem for continuous data have been proposed by Fellegi and Holt (1976), Sande (1978), McKeown (1984), Garfinkel, Kunnathur and Liepins (1988), Ragsdale and McKeown (1996), and Riera-Ledesma and Salazar-González (2003). Software packages for continuous data include GEIS (*cf.* Kovar and Whitridge, 1990) by Statistics Canada, SPEER (*cf.* Winkler and Draper, 1997) by the US Bureau of the Census, AGGIES (*cf.* Todaro, 1999) by NASS, a SAS program developed by the Central Statistical Office of Ireland (*cf.* Central Statistical Office, 2000), and CherryPi (*cf.* De Waal, 1996) by Statistics Netherlands. The latter program is nowadays a module of version 1.0 of SLICE, a general software framework for automatic editing and imputation developed by Statistics Netherlands (*cf.* De Waal, 2001). Algorithms for solving the error localisation problem in a mix of categorical and continuous data are proposed by Sande (1978), Schaffer (1987), De Waal (2003a and 2003b), and De Waal and Quere (2003). In the present article we extend the latter algorithm to a mix of categorical, continuous, and *integer-valued* data. With the exception of De Waal (2003a), part of which formed the basis of the present article, such an algorithm has not yet been described in the literature before.

The remainder of this article is organised as follows. Section 2 sketches the error localisation problem by means of an example. Section 3 describes the edits we consider in this article. Section 4 formulates the error localisation problem as a mathematical optimisation problem. Section 5 sketches the algorithm for solving the error localisation problem in a mix of categorical and continuous data proposed by De Waal and Quere (2003). Section 7 extends this algorithm to a mix of categorical, continuous and integer data. Essential in this extended algorithm is Fourier-Motzkin elimination for

integer data, which we describe in Section 6. This elimination method is due to Pugh (*cf*. Pugh, 1992; Pugh and Wonnacott, 1994), who applied this technique to develop so-called array data dependence testing algorithms. Section 8 discusses a heuristic approach based on the exact algorithm described in Section 7. This heuristic procedure is easier to implement and maintain than the exact algorithm. Computational results for this heuristic procedure are given in Section 9. We conclude the article with a brief discussion in Section 10. In the present article we do not discuss the statistical quality of automatically edited data. For evaluation studies of this aspect of automatic editing we refer to Hoogland and Van der Pijll (2003), and Pannekoek and De Waal (2005).

2 An illustration of the error localisation problem

In this section we illustrate the error localisation problem for a mix of continuous and integer data by means of an example. We also sketch the idea of our solution method for such data, which basically consists of testing whether all integer-valued variables involved in a solution to the corresponding continuous error localisation problem, *i.e*. the error localisation problem where all numerical variables are assumed to be continuous, can indeed attain integer values.

Suppose a set of edits is given by

$$T = P + C, \tag{1}$$

$$0.5T \leq C, \tag{2}$$

$$C \leq 1.1T, \tag{3}$$

$$T \leq 550N, \tag{4}$$

$$320N \leq C \tag{5}$$

$$T \geq 0, \tag{6}$$

$$C \geq 0, \tag{7}$$

$$N \geq 0, \tag{8}$$

where T denotes the turnover of an enterprise, P its profit, C its costs, and N the number

of employees. The turnover, profit and costs are continuous variables, the number of employees an integer one. The original edits, (1) to (8) in this case, are called the *explicit* edits.

Let us consider a specific record with values T = 5,060, P= 2,020, C = 3,040 and N = 5. This record fails edit (4). We apply the Fellegi-Holt paradigm, and try to make the record satisfy all edits by changing as few variables as possible. As T and N occur in the failed edit, it might be possible to satisfy all edits by changing the value of one of these variables only. However, if we were to change the value of T, we would also need to change the value of P or C in order not to violate (1). We therefore start by considering the option of changing N. We first treat N as a continuous variable. To test then whether N can be changed so that all edits (1) to (8) become satisfied, we eliminate N by means of Fourier-Motzkin elimination (*cf.* Duffin, 1974; Chvátal, 1983; Schrijver, 1986; see also Section 5 of the present article). We combine all upper bounds on N (in this case only (5)) with all lower bounds on N (in this case (4) and (8)), to eliminate N from these edits. We obtain a new constraint, given by

$$320T \leq 550C \quad \text{(combination of (4) and (5))} \tag{9}$$

An edit such as (9) that is implied by the original set of edits is called an *implicit*, or *implied*, edit.

The constraints not involving N, *i.e.* (1), (2), (3), (6), (7) and (9) are all satisfied by the original values of T, P and C. A fundamental property of Fourier-Motzkin elimination is that a set of (in)equalities can be satisfied if and only if the set of (in)equalities after the elimination of a variable can be satisfied. This implies that the edits (1) to (8) can be satisfied by changing the value of N only. That is, if N were continuous, the (only) optimal solution to the above error localisation problem would be: change the value of N. However, N is an integer-valued variable. So, we need to test whether a feasible *integer* value for N exists. By filling in the values for T, P, and C in (4) and (5) we find $9.2 \leq N \leq 9.8$. In other words, a feasible integer value for N does not exist. Changing the value of N is hence not a solution to this error localisation problem.

The next best solution to the continuous error localisation problem is given by: change the values of T, P and C (see Section 5 for an algorithm to obtain this solution). This is obviously also a feasible solution to the error localisation problem for continuous and integer data under consideration, as in this solution variable N retains its original value, *i.e.* 5, which is integer. It is the (only) optimal solution to our problem as this is the best solution to the corresponding continuous error localisation problem for which all integer-valued variables can indeed attain integer values.

In this example it is quite easy to check whether a solution to the continuous error localisation problem is also a solution to the error localisation problem for continuous and integer data. In general, this is not the case, however. In Sections 6 and 7 we describe in detail how to test whether integer variables involved in a solution to the continuous error localisation problem can indeed attain feasible integer values.

3 The edits

3.1 Formal definition of edits

We denote the categorical variables by v_i (i=1,...,m) and the numerical variables by x_j (j=1,...,n). For categorical data we denote the domain, *i.e.* the set of the possible values, of variable i by D_i. I denotes the index set of the integer variables, *i.e.* x_j (j=1,...,n) is an integer-valued variable if and only if $j \in I$ and a continuous variable otherwise. We assume that each edit k (k=1, ..., K) is given by

$$\text{IF } v_i \in F_i^k \text{ (for all } i=1,\ldots,m) \text{ THEN } (x_1,\ldots,x_n) \in \{\mathbf{x} | a_{1k}x_1 + \cdots + a_{nk}x_n + b_k \geq 0\}, \quad (10a)$$

or by

$$\text{IF } v_i \in F_i^k \text{ (for all } i=1,\ldots,m) \text{ THEN } (x_1,\ldots,x_n) \in \{\mathbf{x} | a_{1k}x_1 + \cdots + a_{nk}x_n + b_k = 0\}. \quad (10b)$$

Edit k is satisfied by a record (v_1,..., v_m, x_1,..., x_n) if (10a), respectively (10b) holds true. The a_{jk} (j=1,...,n; k=1,...,K) and the b_j (j=1,...,n) are assumed to be rational numbers. $F_i^k \subseteq D_i$ for all $i$and k. We often refer to edits of type (10a) as *inequality* edits and to edits of type (10b) as *balance* edits.

The condition after the IF-statement, *i.e.* "$v_i \in F_i^k$ (for all i=1,...,m)", is called the IF-condition of the edit. The condition after the THEN-statement is called the THEN-condition. A categorical variable v_i is said to *enter* an edit k given by (10) if $F_i^k \subset D_i$ and $F_i^k \neq D_i$, *i.e.* if F_i^k is strictly contained in the domain of variable i. That edit is then said to be *involved with* this categorical variable. A numerical variable x_j is said to *enter* the THEN-condition of edit k given by (10) if $a_{jk} \neq 0$. That THEN-condition is then said to be *involved with* this numerical variable. By multiplying the a_{jk} (j=1,...,n; k=1,...,K) and the b_j (j=1,...,n) involved in the THEN-condition of the k-th (k=1,...,K) edit of type (10) by an appropriately chosen integer we can ensure that in each THEN-condition these coefficients become integral and that their greatest common divisor in this THEN-condition equals 1. Such an edit is then said to be *normalised*.

If the set in the THEN-condition of (10) is the entire n-dimensional real vector space, then the edit is always satisfied and may be discarded. If the set in the THEN-condition of (10) is empty, then the edit is failed by any record for which the IF-condition holds true, *i.e.* for any record for which $v_i \in F_i^k$ for all i=1,...,m. If a set $F_i^k = \emptyset$ for some i=1,...m, then the edit is by definition satisfied and may be discarded. If the IF-condition of an edit does not hold true for a particular record, the edit is satisfied, irrespective of the values of the numerical variables (provided they are not missing).

All edits given by (10) and all integrality constraints have to be satisfied simultaneously. We assume that the edits and integrality constraints can indeed be satisfied simultaneously. We also assume that for each variable entering the edits a value

has to be filled in. Any field for which the value is missing is hence considered to be erroneous. An edit involved with a missing value is considered failed. Any non-integral value for an integer-valued variable is also considered erroneous.

3.2 Examples of edits

Below we illustrate what kinds of edits can be expressed in the form (10) by means of a number of examples, which are taken from De Waal (2003b).

(i) $Turnover - Profit \geq 0$.

This is an example of a numerical edit. The edit can be formulated in our standard form as:

IF $v_i \in D_i$ (for all i=1,...,m) THEN ($Profit$, $Turnover$) $\in$ {($Profit$, $Turnover$)| $Turnover - Profit \geq 0$}.

In the remaining examples we will be less formal with our notation, as we will omit the terms "$v_i \in D_i$" from the edits.

(ii). IF (*Gender* = *"Male"*) THEN (*Pregnant* = *"No"*).

This is an example of a categorical edit. It can be formulated in our standard form as:

IF ((*Gender* = *"Male"*) AND (*Pregnant* = *"Yes"*)) THEN $\emptyset$.

(iii) IF (*Occupation* = *"Statistician"*) THEN (*Income* $\geq$ 1,000 Euro). (11)

This is an example of a mixed edit. Conditional on certain categorical values, a certain numerical constraint has to be satisfied.

(iv) IF ((*Occupation* = *"Statistician"*) OR (*Education* = *"University"*)) THEN (*Income* $\geq$ 1,000 Euro).

This edit can be split into two edits given by (11) and

IF (*Education* = *"University"*) THEN (*Income* $\geq$ 1,000 Euro).

(v) IF (*Tax on Wages* > 0) THEN (*Number of Employees* ≥ 1). (12)

Edit (12) is not in our standard form (10), because the IF-condition involves a numerical variable. To handle such an edit, we introduce an auxiliary categorical variable *TaxCond* with domain {"*false*", "*true*"} during a pre-processing step. Initially, *TaxCond* is set to "*true*" if *Tax on Wages* > 0 in the unedited record, and to "*false*" otherwise. Its reliability weight is set to zero. We now replace edit (12) by the following three edits of type (10):

IF (*TaxCond* = "*false*") THEN (*Tax on Wages* ≤ 0),
IF (*TaxCond* = "*true*") THEN (*Tax on Wages* $\geq \varepsilon$),
IF (*TaxCond* = "*true*") THEN (*Number of Employees* ≥ 1),

where ε is a sufficiently small positive number. The initial value of *TaxCond* may be considered erroneous by the algorithm proposed in this article.

4 A mathematical formulation of the error localisation problem

In this section we give a mathematical formulation of the error localisation problem for a mix of categorical, continuous and integer data. 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, or more precisely: have to ensure the existence of, a synthetic record $(v_1, \ldots, v_m, x_1, \ldots, x_n)$ such that

$$\sum_{i=1}^{m} w_i^c \delta(v_i^0, v_i) + \sum_{j=1}^{n} w_j^r \delta(x_j^0, x_j) \tag{13}$$

is minimised subject to the conditions that all edits $k = 1,\ldots,K$ of type (10) become satisfied, x_j is integer for $j \in I$, and the remaining x_j are continuous. Here w_i^c is the reliability weight of categorical variable i (i=1,...,m), w_j^r the reliability weight of numerical variable j (j=1,...,n), $\delta(y^0, y) = 1$ if $y^0 \neq y$ or y^0 is missing, and $\delta(y^0, y) = 0$ if $y^0 = y$. The objective function (13) is the weighted number of fields that have to be changed. The variables for which the value in the synthetic record differs from the original value plus the variables for which the original value was missing together form an optimal solution to the error localisation problem. Whenever we refer to the error localisation problem in the remainder of this article, we will mean the above mathematical optimisation problem.

Note that if $w_i^c = 1$ for all i=1,...,n and w_j^r for all j=1,...,m, then minimising the objective function (13) reduces to the original paradigm of Fellegi and Holt (1976). The objective function (13) is the most natural generalisation of the paradigm of Fellegi and Holt.

Our aim is to find several, preferably all, optimal solutions to the error localisation problem. The reason for pursuing this goal instead of finding only one optimal solution is that the actual statistical problem of automatic statistical data editing is more comprehensive than the above optimisation problem. This statistical problem is the problem of obtaining high quality data from a data set with errors in an efficient manner. Statistical aspects, such as the probability distribution of the corrected data, need to be taken into account besides the (weighted) number of fields that need to be modified. By generating several optimal solutions to our optimisation problem, we gain the option to later use a secondary, statistical criterion to select one optimal solution

that is best from a statistical point of view (*cf.* Stoop, 2003). The variables involved in the selected optimal solution are set to missing. They are subsequently imputed during the imputation step, for instance by means of regression imputation or donor imputation (see Kalton and Kasprzyk, 1986, and Kovar and Whitridge, 1995, for an overview of imputation methods). In the present article we will not examine the process of selecting one optimal solution from several optimal solutions, neither will we examine the imputation process.

5 A branch-and-bound algorithm for categorical and continuous data

In this section we sketch the branch-and-bound algorithm for solving the error localisation problem for a mix of categorical and continuous data proposed by De Waal and Quere (2003). We assume for the moment that no values are missing. The algorithm is based on constructing a binary tree. An example of such a tree is given in Figure 1.

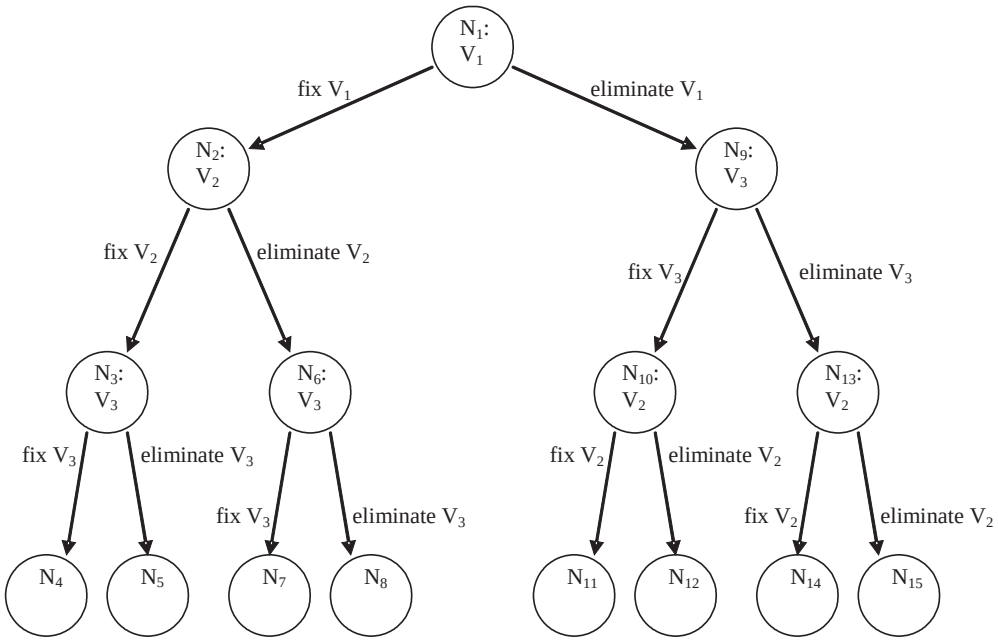

Figure 1: *A binary tree involving three variables.*

In each node of this tree we select a variable that has not yet been selected in any predecessor node. If all variables have been selected, we have reached a terminal node of the tree. After selection of a variable two branches are 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. For instance, in node N_1 of Figure 2 variable V_1

is selected. In the left-hand branch variable V_1 is fixed to its original value, 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). Fixing a variable to its original value corresponds to assuming that this value is correct, 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. In the algorithm, all continuous variables are selected before any categorical variable is. When a variable is fixed or eliminated, the set of current edits is updated. The set of edits corresponding to the root node of our tree is the original set of edits.

We now discuss how to update the set of current edits. We distinguish between fixing and eliminating a variable, and also between categorical and continuous variables. To fix a variable, either continuous or categorical, to its original value we substitute this value in all current edits. Note that, given that we fix this variable to its original value, the new set of current edits is a set of (implicit) edits for the remaining variables in the tree, *i.e.* 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 anymore. 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 amounts to generating a set of implicit edits that do not involve this variable. That set of implicit edits becomes the set of current edits corresponding to the new branch of the tree. If a continuous variable is to be eliminated, we basically apply Fourier-Motzkin elimination (*cf.* Duffin, 1974; Schrijver, 1986) to eliminate that variable from the set of edits. 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 set of current edits, we start by copying all edits not involving x_r from the set of current edits to the new set of edits.

Next, we examine all edits involving x_r pair-wise. Suppose we consider the pair consisting of edits s and t. We start by checking whether the intersection of the IF-conditions is non-empty, *i.e.* whether the intersections $F_i^s \cap F_i^t$ are non-empty for all i=1,...,m. If any of these intersections is empty, we do not consider this particular combination of edits anymore. If all intersections are non-empty, we try to construct an implicit edit. If edit s is a balance edit (10b), we use the equality

$$x_r = -\frac{1}{a_{rs}} \left(b_s + \sum_{j \neq r} a_{js} x_j \right)$$

to eliminate x_r from the THEN-condition of edit t. Similarly, if edit s is an inequality edit (10a) and edit t is a balance equality, the equality in the THEN-condition of edit t is used to eliminate x_r from the THEN-condition of edit s.

If both edits s and t are inequality edits, we check whether the coefficients of x_r in those inequalities have opposite signs, *i.e.* whether $a_{rs} \times a_{rt} < 0$. If the coefficients of x_r in the two inequalities do not have opposite signs, we do not consider this particular combination of edits anymore. If the coefficients of x_r in the two inequalities do have opposite signs, one of the inequalities can be written as a lower bound on x_r and the other as an upper bound on x_r. Combining these two bounds leads to an inequality not involving x_r. To be precise, we generate the THEN-condition:

$$(x_1, \ldots, x_n) \in \{\mathbf{x} | \tilde{a}_1 x_1 + \cdots + \tilde{a}_n x_n + \tilde{b} \geq 0\}, \tag{14}$$

where

$$\tilde{a}_j = |a_{rs}| \times a_{jt} + |a_{rt}| \times a_{js} \quad \text{for all} \quad j = 1, \ldots, n$$

and

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

The above THEN-condition forms the THEN-condition of a new implied edit. Note that x_r indeed does not enter this THEN-condition. The IF-condition of this implicit edit is given by the intersections $F_i^s \cap F_i^t$ for all i=1,...,m. Intuitively it is clear that the IF-condition of the new implicit edit is given by the *intersections* $F_i^s \cap F_i^t$ (i=1,...,m), because two (numerical) THEN-conditions can only be combined into the (numerical) THEN-condition of an implicit edit for the overlapping part of their corresponding categorical IF-conditions. Note that if we eliminate a continuous variable in any of the ways described above, the resulting set of edits is a set of implicit 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. Repeatedly applying the above elimination process until all continuous variables have been eliminated results in edits with two kinds of THEN-conditions, namely edits with a THEN-condition that is trivially true, *e.g.* "$1 \geq 0$", and edits with a THEN-condition that is trivially false, *e.g.* "$0 \geq 1$". The edits with a THEN-condition of the former kind are deleted.

As we already mentioned before, categorical variables are treated, *i.e.* fixed or eliminated, after all continuous variables have been treated. So, when the categorical variables may be selected all edits in the set of current edits have the following form:

$$\mathsf{IF} v_i \in F_i^k \quad (\text{for} \quad i = 1, \ldots, m) \ \mathsf{THEN}(x_1, \ldots, x_n) \in \emptyset. \tag{15}$$

To eliminate categorical variable v_r from the set of edits given by (15), we start by copying all edits not involving v_r to the new set of edits. Next, we basically apply the method of Fellegi and Holt to the IF-conditions to generate the IF-conditions of the new edits (*cf.* Fellegi and Holt, 1976). 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

$$\bigcup_{k \in S} F_r^k = D_r \tag{16}$$

and

$$\bigcap_{k \in S} F_i^k \neq \emptyset \quad \text{for all} \quad i = 1, \ldots, r-1, r+1, \ldots, m. \tag{17}$$

From these index sets we select the *minimal* ones, *i.e.* the index sets S that obey (16) and (17), but none of their subsets obey (16). Given such a minimal index set S we construct the edit given by

$$\text{IF} v_r \in D_r, v_i \in \bigcap_{k \in S} F_i^k \quad (\text{for } i = 1, \ldots, r-1, r+1, \ldots, m) \text{ THEN}(x_1, \ldots, x_n) \in \emptyset.$$

Note that if we eliminate a categorical variable in the way described above, the resulting set of edits is a set of implicit 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.

If values are missing in the original record, the corresponding variables only have to be eliminated (and not fixed) from the set of current edits. These variables are considered erroneous, and have to be imputed.

We have now explained how the set of current edits changes if we fix or eliminate a variable. 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

$$\text{IF } v_i \in D_i \quad (\text{for } i = 1, \ldots, m) \text{ THEN } (x_1, \ldots, x_n) \in \emptyset.$$

We have the following theorem.

Theorem 1 *A set of relations obtained after all categorical variables have been treated 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. modified such that all original edits can be satisfied.*

Theorem 1 follows from a repeated application of the following theorem, which is proved in De Waal and Quere (2003), and the fact that eliminating a variable amounts to generating a set of implied edits for the remaining variables

Theorem 2 *Suppose that the index set of the variables in a certain node is given by T_0, and the set of current edits corresponding to that node by Ω_0. Suppose furthermore that*

to obtain a next node 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.

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 sets of variables that can be imputed consistently we select the ones with the lowest sums of reliability weights. In this way we find all optimal solutions to the error localisation problem.

The 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, balance edits can often be handled more efficiently than we described here (*cf.* De Waal and Quere, 2003; De Waal, 2003a). Second, if the minimum number of fields that have to be changed in order to make a record pass all edits is large, we feel that the record should not be edited automatically. In our opinion, the quality of such a record is too low to correct it automatically. We suggest that such a record should either be edited manually, or be discarded completely. This is similar to selective editing (*cf.* Lawrence and McKenzie, 2000; Hedlin, 2003) where (only) the very influential and very contaminated records are selected for manual correction. Such very influential and very contaminated records can be corrected by using subject-matter knowledge or, in the worst case, by re-contacting the supplier of the data. By specifying an upper bound on the number of fields that may be changed, the size of the tree can drastically be reduced. Third, the size of the tree can also be reduced during the execution of the algorithm, because it may already become clear in an intermediate 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 current edits infeasible, which may be noticed in an intermediate node. Fourth, the value of the objective function can be used as an incumbent in order to reduce the size of the tree. This value cannot decrease while going down the tree. So, if the objective value 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. In other words, the objective value of the best already found solution is used as the bound in our branch-and-bound scheme. During the execution of the algorithm the bound is updated.

In their article Fellegi and Holt (1976) describe a method for solving the error localisation problem that is also based on generating implicit edits. They propose to generate a, generally very large, set of implicit edits, which they refer to as the *complete set of edits*, for all records simultaneously before the error localisation problem is actually solved for each record. Given this "complete" set of edits, the error localisation problem can, for each record, be formulated as a set-covering problem (see, *e.g.*, Nemhauser and Wolsey, 1988, for more on the set-covering problem in general). In

contrast, in our algorithm we generate implicit edits while solving the error localisation problem for each record separately. The implicit edits are generated conditional on which variables have been selected for elimination and on the observed data in the record under consideration. As a result, our sets of implicit edits are generally much smaller than the "complete" set of edits generated by the approach of Fellegi and Holt. Over the years, generating this "complete" set of edits has often proven to be infeasible for large or even moderately-sized problems (*cf.* Winkler, 1996). This has been confirmed by some earlier experiments at Statistics Netherlands. Generating the smaller sets of implicit edits in our algorithm has turned out to be feasible for most records arising in practice. We refer to De Waal and Quere (2003) and Section 9 of the present article for confirmation of this assertion.

6 Fourier-Motzkin elimination in integer data

An important technique used in the algorithm described in Section 5 is Fourier-Motzkin elimination for eliminating a continuous variable from a set of linear (in)equalities. Fourier-Motzkin elimination can be extended to integer data in several ways. For example, Dantzig and Eaves (1973) and Williams (1976 and 1983) describe extensions of Fourier-Motzkin elimination to integer programming problems. Unfortunately, these methods seem too time-consuming in many practical cases. Pugh (1992) proposes an alternative extension that he refers to as the Omega test. Pugh (1992) and Pugh and Wonnacott (1994) claim a good performance of this test for many practical cases. Below we briefly explain the Omega test. For more details we refer to Pugh (1992), and Pugh and Wonnacott (1994).

The Omega test has been designed to determine whether an integer-valued solution to a set of linear (in)equalities exists. Suppose linear (in)equality k (k=1,...,K) is given by

$$a_{1k}x_1 + \cdots + a_{nk}x_n + b_k \geq 0,$$

or by

$$a_{1k}x_1 + \cdots + a_{nk}x_n + b_k = 0.$$

To simplify our notation we define $x_0 = 1$ and $a_{0k} = b_k$ (k=1,...,K), and re-write the above linear (in)equality as

$$a_{0k}x_0 + a_{1k}x_1 + \cdots + a_{nk}x_n \geq 0, \tag{18a}$$

respectively as

$$a_{0k}x_0 + a_{1k}x_1 + \cdots + a_{nk}x_n = 0. \tag{18b}$$

Without loss of generality we assume that redundant equalities have been removed, and that all (in)equalities are normalised, *i.e.* that all a_{jk} (j=0,...,n; k=1,...,K) are integer and the greatest common divisor of the a_{jk} in each constraint k equals 1. All variables x_j (j=0,...,n) are integer-valued in this section.

We start by "eliminating" all equalities until we arrive at a new problem involving only inequalities. In this context, we say that all equalities have been eliminated once we have transformed the original system of (in)equalities (18) into an equivalent system of (in)equalities of the following type:

$$x'_k = \sum_{j>k} a'_{jk} x'_j \quad \text{for } k = 0, \ldots, s-1, \tag{19a}$$

$$\sum_{j \geq s} a'_{jk} x'_j \geq 0 \quad \text{for } k = s, \ldots, K', \tag{19b}$$

where s is the number of equalities in the system (19), and the a'_{jk} are integer. The x'_j are a permutation of the x_j, possibly supplemented by some additional, auxiliary variables (see Subsection 6.1). We call a set of (in)equalities (18) equivalent to a set of (in)equalities (19) if a solution to the system (18) can be extended to a corresponding solution to the system (19), and conversely a solution to the system (19) is also a solution to the system (18) if we disregarded the additional variables. In (19), the first s x'_j, which are only involved in equalities, are expressed in terms of the remaining variables, which may also be involved in inequalities. Owing to the possible introduction of additional variables, the system (19) may have more equalities than the original system (18), so $K' \geq K$. The original system (18) has an integer-valued solution if and only if the system (19b) has an integer-valued solution. Namely, an integer solution for the x'_j ($j \geq s$) to the system (19b) yields an integer solution to the system (19), *i.e.* the system consisting (19a) plus (19b), by applying back-substitution to the x'_j ($j < s$). In other words, to check whether a system (18) has an integer-valued solution, we only need to check whether the inequalities (19b) of the equivalent system (19) have an integer-valued solution. In this sense the equalities of (18) have been eliminated once we have transformed a system given by (18) into an equivalent system given by (19).

6.1 Eliminating equalities

We now discuss how to eliminate an equality. As usual we denote the number of numerical, in this subsection: integer-valued, variables by n. We define the operation $c \overline{\bmod} d$ involving two integers c and d by

$$c \overline{\bmod} d = c - d \lfloor c/d + 1/2 \rfloor, \tag{20}$$

where $\lfloor u \rfloor$ denotes the largest integer less than or equal to u. If d is odd, the value of $c \overline{\bmod} d$lies in $[-(d-1)/2, (d-1)/2]$. If d is even, the value of $c \overline{\bmod} d$ lies in

$[-d/2, d/2-1]$. If $\lfloor c/d \rfloor < 1/2$, then $c \overline{\bmod} d = c \mod d$. If $\lfloor c/d \rfloor \geq 1/2$, then $c \overline{\bmod} d = -c \mod d$. Here, the mod d operator assumes values in [0, d-1].

To eliminate an equality s given by

$$\sum_{j=0}^{n} a_{js} x_j = 0, \tag{21}$$

we select an r such that $a_{rs} \neq 0$ and $|a_{rs}|$ has the smallest value among the a_{js} (j=0,...,n). If $|a_{rs}| = 1$, we eliminate the equality by using this equality to express x_r in terms of the other variables, and substitute this expression for x_r into the other (in)equalities. Otherwise, we define $\gamma = |a_{rs}| + 1$. Now we introduce a new variable σ defined by

$$\gamma\sigma = \sum_{j=0}^{n} (a_{js} \overline{\bmod} \gamma)\, x_j. \tag{22}$$

This variable σ is integer-valued. This can be shown as follows.

$$\sum_{j=0}^{n} (a_{js} \overline{\bmod} \gamma)\, x_j = \sum_{j=0}^{n} \left(a_{js} - \gamma \left\lfloor a_{js} \middle/ \gamma + 1/2 \right\rfloor \right) x_j = -\sum_{j=0}^{n} \gamma \left\lfloor a_{js} \middle/ \gamma + 1/2 \right\rfloor x_j, \tag{23}$$

where we have used (21). So, σ equals $-\sum_{j=0}^{n} \left\lfloor a_{js} \middle/ \gamma + 1/2 \right\rfloor x_j$, which is integer because the x_j (j=0,...,n) and their coefficients in (23) are integer.

It is easy to see that $a_{rs} \overline{\bmod} \gamma = -\text{sign}(a_{rs})$. Now, we use (22) to express x_r in terms of the other variables.

$$x_r = -\text{sign}(a_{rs}) q\sigma + \sum_{j=0, j\neq r}^{n} \text{sign}(a_{rs})(a_{js} \overline{\bmod} q)\, x_j \tag{24}$$

Substituting (24) into the original equality (21) gives

$$-|a_{rs}|\, \gamma\sigma + \sum_{j=0, j\neq r}^{n} (a_{js} + |a_{rs}|\, (a_{js} \overline{\bmod} \gamma))\, x_j = 0. \tag{25}$$

Because $|a_{rs}| = \gamma - 1$, (25) can be written as

$$-|a_{rs}|\, \gamma\sigma + \sum_{j=0, j\neq r}^{n} (a_{js} - (a_{js} \overline{\bmod} \gamma) + \gamma(a_{js} \overline{\bmod} \gamma))\, x_j = 0 \tag{26}$$

Using (20) on (26), and dividing by γ gives

$$-|a_{rs}|\,\sigma + \sum_{j=0, j\neq r}^{n} \left(\left\lfloor a_{js} \big/ \gamma + 1/2 \right\rfloor + (a_{js} \overline{\bmod}\ \gamma) \right) x_j = 0. \tag{27}$$

In (27) all coefficients are integer-valued.

It is clear that if the coefficient of variable x_j (j=0,...,n) equals zero in (21), the corresponding coefficient in (27) also equals zero. It is also clear that the absolute value of the coefficient of σ in (27) is equal to the absolute value of the coefficient of x_r in (21). However, for all other variables with a non-zero coefficient in (21) the absolute value of the corresponding coefficient in (27) is smaller than the absolute value of the coefficient in (21). To prove this statement we first re-write the coefficient of x_j ($j \neq r$) in (27) in the following way:

$$\left\lfloor a_{js} \big/ \gamma + 1/2 \right\rfloor + (a_{js} \overline{\bmod}\ \gamma) = \left\lfloor a_{js} \big/ \gamma + 1/2 \right\rfloor + a_{js} - \gamma \left\lfloor a_{js} \big/ \gamma + 1/2 \right\rfloor =$$

$$-|a_{rs}| \left\lfloor \frac{a_{js}}{|a_{rs}|+1} + \frac{1}{2} \right\rfloor + a_{js} \equiv \hat{a}_{js},$$

where we have used again that $\gamma = |a_{rs}| + 1$. We now consider the cases where a_{js} is positive and negative separately. If $a_{js} > 0$, then $a_{js} \geq |a_{rs}|$ by our choice of r. Suppose $a_{js} = \lambda\ |a_{rs}|$, where $\lambda \geq 1$. We then have

$$\hat{a}_{js} = a_{js} \left(-\frac{1}{\lambda} \left\lfloor \frac{\lambda\,|a_{rs}|}{|a_{rs}|+1} + \frac{1}{2} \right\rfloor + 1 \right).$$

Using

$$1 \leq \left\lfloor \frac{\lambda\,|a_{rs}|}{|a_{rs}|+1} + \frac{1}{2} \right\rfloor \leq \lambda$$

for all possible values of $|a_{rs}|$, we obtain $0 \leq \hat{a}_{js} \leq (1 - \frac{1}{\lambda}) a_{js}$. Hence, we can conclude that $\left|\hat{a}_{js}\right| < \left|a_{js}\right|$. In a similar way, one can show that if $a_{js} < 0$, then too $\left|\hat{a}_{js}\right| < \left|a_{js}\right|$. This is left for the reader to verify.

After a repeated application of the above substitution rule, where each time a new variable is introduced and an old variable is eliminated, to the original equality (21) and its derived form(s) (27), the equality is transformed into an equality in which (at least) one of the coefficients has absolute value 1. The corresponding variable can then be expressed in terms of the other variables. We substitute this expression into the other (in)equalities. The equality has then been eliminated.

This process continues until we have eliminated all equalities and we have obtained a system of the form (19). In the next subsection we explain how integer variables can

be eliminated from a set of linear inequalities (19b), but first we give an example of how equalities are eliminated.

Example 1 *We repeat part of an example given by Pugh (1992). In this example, four constraints have been specified:*

$$7x + 12y + 31z = 17, \tag{28}$$

$$3x + 5y + 14z = 7, \tag{29}$$

$$1 \leq x \leq 40, \tag{30}$$

Note that (30) stands for two inequalities. We wish to eliminate equality (28). Note that $\gamma = 8$*, and using (22) we introduce a variable* σ *defined by*

$$8\sigma = -x + 4y + z + 1. \tag{31}$$

We eliminate x from (28) to (30). Applying rule (27) to constraint (28) yields

$$-7\sigma - 2y + 3z = 3, \tag{32}$$

and applying rule (24) to constraints (29) and (30) yields

$$-24\sigma - 7y + 11z = 10, \tag{33}$$

$$1 \leq -8\sigma - 4y - z - 1 \leq 40. \tag{34}$$

The absolute values of the coefficients of y and zin (32) are smaller than the absolute values of the corresponding coefficients in (28). The system (31) to (34) is equivalent to the system (28) to (30).

6.2 Eliminating an integer variable from a set of inequalities

When an integer variable is eliminated from a set of inequalities involving only integer-valued variables, two different regions are determined. The first region is referred to as the *real shadow*. This is simply the region described by the set of inequalities that results if we apply the standard form of Fourier-Motzkin elimination. That is, the real shadow

results if we treat the integer variable that is being eliminated as continuous. The second region is referred to as the *dark shadow*. This dark shadow is constructed in such a way that if it contains a feasible (integer) solution, then the existence of a feasible (integer) solution to the original inequalities is guaranteed.

We describe the construction of the dark shadow. Suppose that two inequalities

$$ax \leq \alpha \tag{35}$$

and

$$bx \geq \beta \tag{36}$$

are combined to eliminate the integer variable x. Here a and b are positive integer constants, and α and β are linear expressions that may involve all variables except x. Each variable involved in α or β is assumed to have an integer coefficient. The real shadow obtained by eliminating x from the pair of inequalities (35) and (36) is defined by

$$a\beta \leq b\alpha. \tag{37}$$

We define the real shadow obtained by eliminating a variable $x$from a set of inequalities S to be the region described by the inequalities in S not involving x, and the inequalities (37) generated by all pairs of upper bounds (35) on x and lower bounds (36) on $x$in S.

Now, consider the case in which there is an integer value larger than or equal to $a\beta$ and smaller than or equal to $b\alpha$, but there is no integer solution for x to $a\beta \leq abx \leq b\alpha$. Let $q = \lfloor \beta/b \rfloor$, then by our assumptions we have

$$abq < a\beta \leq b\alpha < ab(q+1).$$

We clearly have $a(q+1) - \alpha > 0$. Since the values of a, b, α and β are integer, we have $a(q+1) - \alpha \geq 1$, and hence

$$ab(q+1) - b\alpha \geq b. \tag{38}$$

Similarly, we obtain

$$a\beta - abq \geq a. \tag{39}$$

Combining (38) and (39), we arrive at

$$b\alpha - a\beta \leq ab - a - b.$$

In other words, if

$$b\alpha - a\beta \geq ab - a - b + 1 = (a-1)(b-1), \tag{40}$$

then an integer solution for x necessarily exists.

To be able to satisfy (35) and (36) by choosing an appropriate integer value for x it is sufficient that (40) holds true. We therefore define the dark shadow obtained by eliminating variable x from the pair of inequalities (35) and (36) by the region described by (40). Note that if (40) holds true, there is an integer value larger than or equal to $a\beta$ and smaller than or equal to $b\alpha$. We define the dark shadow obtained by eliminating a variable $x$from a set of inequalities S to be the region described by the inequalities in S not involving x, and the inequalities (40) generated by all pairs of upper bounds (35) on x and lower bounds (36) on $x$in S.

We now consider a set of inequalities S with only integer-valued coefficients and variables. If the real shadow and the dark shadow resulting from the elimination of x from S are identical, we say that the elimination, or projection, is *exact*. In that case, an integer solution exists if and only if an integer solution to the real/dark shadow exists. If the real shadow and the dark shadow are not identical, we have the following possibilities:

- If the dark shadow has an integer solution, the set of inequalities S has an integer solution.
- If the real shadow does not contain a feasible (integer) solution, there is no integer solution to the set of inequalities S.
- In all other cases, it is not yet clear whether an integer solution to the set of inequalities S exists.

In the latter case we know that if an integer solution to the set of inequalities S were to exist, a pair of constraints $ax \leq \alpha$ and $\beta \leq bx$ would exist such that $ab-a-b \geq b\alpha - a\beta$ and $b\alpha \geq abx \geq a\beta$. From this we can conclude that in such a case an integer solution to the set of inequalities S would satisfy $ab - a - b + a\beta \geq abx \geq a\beta$. We can check whether an integer solution to the set of inequalities S exists by examining all possibilities. Namely, we determine the largest coefficient a_{max} of x for all upper bounds (35) on x. For each lower bound $\beta \leq bx$ we then test whether an integer solution exists to the original constraints S combined with $bx = \beta + p$ for each integer p satisfying $(a_{\max}b - a_{\max} - b)/a_{\max} \geq p \geq 0$. That is, in the latter case we examine $\lfloor (a_{\max}b - a_{\max} - b)/a_{\max} \rfloor + 1$ subproblems of the original problem. These subproblems are referred to as *splinters*.

The theory discussed so far shows that if the dark shadow or one of the splinters has an integer solution, then the original set of inequalities S has an integer solution. Conversely, because we examine all possibilities, it also holds true that if the original set of inequalities S has an integer solution then the dark shadow or one of the splinters has an integer solution. So, we have demonstrated the following theorem.

Theorem 3 *If and only if an integer solution to the dark shadow or one of the splinters exists, then an integer solution to the original set of inequalities S exists.*

Note that if the original set of inequalities S involves n integer variables, the dark shadow and the splinters involve only n-1 integer variables (for the splinters the added equality $bx = \beta + p$ first has to be eliminated in order to arrive at a system of inequalities involving n-1 variables). We have now explained how we can check whether a feasible integer value exists for an integer variable involved in a set of linear inequalities by eliminating this variable. In the next subsection we examine how we can test whether an integer solution exists for several variables simultaneously by eliminating these variables.

6.3 Eliminating several integer variables from a set of inequalities

Suppose we want to determine whether an integer solution exists for a set of linear inequalities involving n variables. We solve this problem by eliminating these n variables. During the elimination process the original problem may split into several subproblems owing to the splinters that may arise. We apply the procedure sketched below. We focus on the idea underlying the procedure; the computational efficiency of the procedure is ignored here.

We construct a list of subproblems. At the start of the procedure the only (sub)problem is the original problem involving all n variables. We treat each subproblem that may arise separately. We now consider one of those subproblems. We eliminate all variables involved in this subproblem by means of standard Fourier-Motzkin elimination, *i.e.* we repeatedly determine the real shadow until all variables have been eliminated. If the final real shadow without any unknowns is inconsistent, the subproblem does not have a continuous solution, let alone an integer solution. In such a case this subproblem can be discarded.

If the final real shadow of a subproblem is consistent, and a continuous solution hence exists, we examine the subproblem again and test whether there is an integer solution to this subproblem. For this subproblem we iteratively select a variable from the set of variables that have not yet been eliminated. The selected variable will be eliminated, using the method of Subsection 6.2. In order to keep the number of computations limited we choose the variable so that the elimination will be exact if possible. As a secondary aim we may then also minimise the number of constraints resulting from the combination of upper and lower bounds. If an exact elimination is

not possible, we select a variable with coefficients as close as possible to zero. For such a variable the number of splinters will be relatively small. Testing all splinters for integer solutions can be quite time-consuming, so creating splinters and testing them for integer solutions should be avoided as much as possible. For the subproblem under consideration, we determine the dark shadow and the splinters (if any) by eliminating the selected variable, using the method of Subsection 6.2. The dark shadow and the splinters define new subproblems, and are added to the list of subproblems. After this, we have dealt with the subproblem under consideration, and it is deleted from the list of subproblems. We continue this process until all variables have been eliminated from all subproblems on the list of subproblems. The final "subproblems", or better: final sets of relations, involve only numbers and no unknowns. As in the continuous case (see Section 5), such a relation can be self-contradicting, *e.g.* "$0 \geq 1$". We have the following theorem.

Theorem 4 *If any of the final sets of relations does not contain a self-contradicting relation, the original set of inequalities has an integer solution. Conversely, if all final sets of relations contain a self-contradicting relation, the original set of inequalities does not have an integer solution.*

Proof. This follows from a repeated application of Theorem 3. □

7 Error localisation in categorical, continuous and integer data

In this section we integrate the Omega test described in Section 6 with the branch-and-bound approach for solving the error localisation problem for categorical and continuous data proposed by Quere and De Waal (2003) (see Section 5). The result of this integration is an algorithm for solving the error localisation problem for categorical, continuous, and integer-valued data. The idea of this algorithm is to test whether the integer-valued variables involved in a solution to the continuous error localisation problem, *i.e.* the error localisation problem where all numerical variables are assumed to be continuous, can attain integer values. This is illustrated in Figure 2 below.

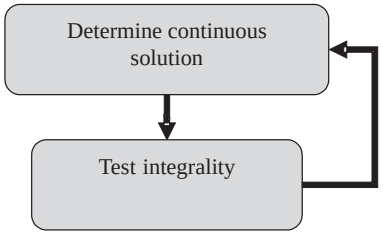

Figure 2: *The basic idea of the error localisation algorithm.*

For a given combination of categorical values, our integrality test reduces to the Omega test. In other words, we basically apply the Omega test on each possible combination of categorical values. What complicates the issue is that we do not explicitly enumerate and test all possible combinations of categorical values. Before we describe the algorithm, we first explain in Subsection 7.1 how balance edits involving integer variables can be "eliminated" and in Subsection 7.2 how integer variables can be eliminated from inequality edits. Finally, Subsection 7.3 describes our algorithm for solving the error localisation problem for categorical, continuous and integer data.

As usual, the edits are given by (10). For notational convenience, we define $x_0 = 1$ and $a_{0k} = b_k$ for k=1,...,K, where K is the number of edits, like we also did in Section 6.

7.1 Error localisation: eliminating balance edits involving integer variables

In our integrality test (see Subsection 7.3) integer variables are treated after all continuous variables have been treated and before any categorical variable is treated. That is, once the integer variables are treated all edits involve only categorical and integer variables. If integer variables are involved in balance edits, we first "eliminate" these edits. We select a balance edit, and basically apply the technique explained in Subsection 6.1 to arrive at an equality in which the absolute value of the coefficient of an integer variable equals 1. During this process the IF-condition of the edit under consideration does not alter. To be more precise, if the selected edit s is given by

$$\text{IF} v_i \in F_i^s \quad (\text{for } i = 1, \ldots, m) \quad \text{THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} | \sum_{j=0}^{n} a_{js} x_j = 0\}, \tag{41}$$

with the a_{js} (j=0,...,n) integer coefficients and the x_j (j=0,...,n) integer variables, we transform this edit into

$$\text{IF } v_i \in F_i^s \quad (\text{for } i = 1, \ldots, m) \text{ THEN } (\tilde{x}_1, \ldots, \tilde{x}_{\tilde{n}}) \in \{\tilde{\mathbf{x}} | \sum_{j=0}^{\tilde{n}} \tilde{a}_{js} \tilde{x}_j = 0\}, \tag{42}$$

where the $\tilde{a}_{js}$ (j=0,...,$\tilde{n}$) are integer coefficients, and the $\tilde{x}_j$ (j=0,..., $\tilde{n}$) are the transformed integer variables, possibly supplemented by some auxiliary integer variables owing to the elimination of the equality. The total number of variables $\tilde{x}_j$ is denoted by $\tilde{n}$ ($\tilde{n} \geq n$). In (42), at least one integer variable, say $\tilde{x}_r$, has a coefficient $\tilde{a}_{rs}$ with $|\tilde{a}_{rs}| = 1$. Below we describe the procedure to transform (41) into (42). For notational convenience, we write $\tilde{n}$ again as n. Likewise, we write the transformed coefficients $\tilde{a}_{js}$ (j=0,..., $\tilde{n}$) and transformed variables $\tilde{x}_j$ (j=0,..., $\tilde{n}$) again as a_{js} and x_j. It is important to keep in mind, though, that these coefficients and variables may differ from the original coefficients and variables.

Because auxiliary variables may need to be introduced during the elimination process of a balance edit, we may in fact need to introduce some auxiliary balance edits of which the THEN-conditions are given by equations of type (22) (or equivalently: of type (24)), and the IF-conditions by the IF-condition of the selected edit s. In each of these auxiliary equations, the new auxiliary variable is expressed in terms of the other integer variables x_j (j=1,...,n), *i.e.* the original integer variables and the already generated auxiliary variables. The other edits are written in terms of the new auxiliary variable by applying the substitution (24) to the numerical THEN-conditions as far as this is permitted by the IF-conditions. The IF-conditions of these other edits are changed by the substitution process. In particular, an edit t given by

$$\text{IF } v_i \in F_i^t \quad (\text{for } i = 1, \ldots, m) \text{ THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} | \sum_{j=0}^{n} a_{jt} x_j \geq 0\}, \tag{43}$$

involving x_r in its THEN-condition gives rise to (at most) two edits given by

$$\begin{aligned} &\text{IF } v_i \in F_i^t \cap F_i^s \quad (\text{for } i = 1, \ldots, m) \\ &\text{THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} | -\text{sign}(a_{rs}) a_{rt} \gamma\sigma + \sum_{j=0, j\neq r}^{n} (a_{jt} + sign(a_{rs}) a_{rt} (a_{js} \overline{\text{mod}}\ \gamma)) x_j \geq 0\}, \end{aligned} \tag{44}$$

and

$$\begin{aligned} &\text{IF } v_i \in F_i^t - F_i^s \quad (\text{for } i = 1, \ldots, m) \\ &\text{THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} | \sum_{j=0}^{n} a_{jt} x_j \geq 0\}. \end{aligned} \tag{45}$$

In (43) to (45) the inequality sign may be replaced by an equality sign. Edits of type (44) for which $F_i^t \cap F_i^s = \emptyset$ (for some i=1,...,m), and edits of type (45) for which $F_i^t - F_i^s = \emptyset$ (for some i=1,...,m) may be discarded. Edits given by (43) not involving x_r are not modified.

Once we have obtained an edit of type (42) with a coefficient a_{rs} such that $|a_{rs}| = 1$, we use the THEN-condition of this edit to express the variable x_r in terms of the other variables. That is, we use

$$x_r = -sign(a_{rs}) \sum_{j=0, j\neq r}^{n} a_{js} x_j. \tag{46}$$

This expression for x_r is then substituted into the THEN-conditions of the other edits as far as this is permitted by the IF-conditions. The IF-conditions of these other edits are changed by the substitution process. In particular, owing to this substitution process an edit given by (43) involving x_r in its THEN-condition gives rise to (at most) two edits

given by (45) and

$$\text{IF } v_i \in F_i^t \cap F_i^s \text{ (for } i = 1, \ldots, m)$$
$$\text{THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} | \sum_{j=0, j\neq r}^{n} (a_{jt} - sign(a_{rs}) a_{rt} a_{js}) x_j \geq 0\}, \quad (47)$$

In (43), (45), and (47) the inequality sign may be replaced by an equality sign. Edits of type (47) for which $F_i^t \cap F_i^s = \emptyset$ (for some i=1,...,m), and edits of type (45) for which $F_i^t - F_i^s = \emptyset$ (for some i=1,...,m) may be discarded. Edits given by (43) not involving x_r are not modified.

The new system of edits is equivalent to the original system of edits, in the sense that a solution to the original system of edits corresponds to a solution to the new system, and vice versa. Namely, for the categorical values for which we can use equation (24) or (46) to eliminate variable x_r, we do this (see (44) and (47)). For the categorical values for which we cannot use equation (24) or (46) to eliminate x_r, we simply leave x_r untouched (see (45)). Note that the IF-conditions of an edit of type (44) or (47) where x_r has been eliminated and an edit still involving x_r have an empty overlap. An edit of type (44) or (47) and an edit still involving x_r will hence never be combined when eliminating integer variables from inequality edits (see Subsection 7.2 for the elimination of integer variables from inequality edits).

We continue "eliminating" balance edits until for each possible combination of categorical values the associated set of numerical THEN-conditions is either a system of type (19) or the empty set. The latter possibility occurs if a combination of categorical values is not allowed by the edits. Note that the balance edits will be eliminated after finitely many steps. Namely, for each possible combination of categorical values we in fact implicitly apply the elimination process of Subsection 6.1, which terminates after a finite number of steps.

After the termination of the above elimination process, we delete all balance edits. We are then left with a set of edits with linear inequalities involving only integer variables as THEN-conditions. Because auxiliary variables may have been introduced to eliminate the balance edits, the total number of integer variables in this system of edits may be larger than the original number of integer variables. How we deal with a set of inequality edits involving only integer variables is explained in the next subsection.

7.2 Error localisation: eliminating integer variables from inequality edits

In this subsection we assume that each THEN-condition is either a linear inequality involving only integer variables or the empty set. When an integer variable is eliminated from a set of inequality edits, a dark shadow and possibly several splinters are generated. Below we describe how this dark shadow and these splinters are defined. We start by selecting an integer variable that we want to eliminate, say x_r. The current edits involving x_r are combined into implicit edits not involving x_r. We consider all edits

involving x_r pair-wise. Such a pair of edits is given by

$$\text{IF } v_i \in F_i^s \quad (\text{for } i = 1, \ldots, m) \text{ THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} \mid \sum_{j=0}^{n} a_{js}\, x_j \geq 0\} \tag{48}$$

and

$$\text{IF } v_i \in F_i^t \text{ (for } i = 1, \ldots, m) \text{ THEN } (x_1, \ldots, x_n) \in \{\mathbf{x} | \sum_{j=0}^{n} a_{jt} x_j \geq 0\}, \tag{49}$$

where all involved numerical variables are integer-valued. We assume that the a_{js}, respectively the a_{jt}, (j=0,...,n) are normalised.

The real shadow obtained by eliminating x_r from the pair of edits (48) and (49) is defined only if $a_{rs} \times a_{rt} < 0$. Its THEN-condition is then given by (14), and its IF-condition by $v_i \in F_i^t \cap F_i^s$ (for i=1,...,m) . The dark shadow is also only defined if $a_{rs} \times a_{rt} < 0$. In that case one coefficient is larger than zero, say $a_{rs} > 0$, and the other coefficient is less than zero, $a_{rt} < 0$. The dark shadow obtained by eliminating x_r from the pair of edits (48) and (49) is then defined by

$$\begin{aligned} &\text{IF } v_i \in F_i^s \cap F_i^t \text{ (for } i = 1, \ldots, m) \\ &\text{THEN} \mathbf{x} \in \{\mathbf{x} | \sum_{j=0}^{n} (a_{rs} a_{jt} - a_{rt} a_{js}) x_j \geq (a_{rs} - 1)(-a_{rt} - 1)\}. \end{aligned} \tag{50}$$

If $F_i^t \cap F_i^s$ is empty for some i=1, ..., m, edit (50) is deleted. As for the real shadow, the IF-condition of the dark shadow (50) is given by the intersections $F_i^s \cap F_i^t$ (i=1,...,m), because two numerical THEN-conditions can only be combined into an implicit numerical THEN-condition for the overlapping parts of their corresponding categorical IF-conditions. Note that for this overlapping part the THEN-condition of the dark shadow is given by (40). The dark shadow obtained by eliminating x_r from a set of inequality edits is by definition given by the edits not involving x_r plus the dark shadows (50), assuming they exist, for all pairs of edits (48) and (49).

Defining the splinters obtained by eliminating x_r from a set of inequality edits is more complicated than in Section 6. The reason is that here we want to define splinters for different combinations of categorical values simultaneously, whereas Section 6 considers the case without any categorical variables. We describe one possibility to define splinters; for an alternative possibility we refer to De Waal (2003a). We write the inequality edits involving x_r as

$$\text{IF } v_i \in F_i^k \quad (\text{for } i = 1, \ldots, m) \text{ THEN } a_{rk} x_r \geq - \sum_{j=0, j \neq r}^{n} a_{jk} x_j. \tag{51}$$

For negative coefficients a_{rk}, the THEN-condition of (51) provides an upper bound on x_r. For positive coefficients a_{rk}, the THEN-condition of (51) provides a lower bound on x_r. We start by determining the smallest negative coefficient a_{rq} of x_r for all edits (51), *i.e.* a_{rq} is the coefficient of x_r in all upper bounds on x_r with the largest absolute value. For each lower bound on x_r, we then test whether an integer solution exists to the original edits combined with

$$\text{IF } v_i \in F_i^k \quad (\text{for } i = 1, \ldots, m) \text{ THEN } a_{rk} x_r = -\sum_{j \neq r} a_{jk} x_j + p \tag{52}$$

for each integer p satisfying $(-a_{rq}a_{rk} + a_{rq} - a_{rk}) \Big/ (-a_{rq}) \geq p \geq 0$. For each possible combination of categorical values, all splinters required according to the Omega test described in Subsection 6.2 are taken into consideration. For some combinations of categorical values, more splinters than necessary are taken into consideration. These superfluous splinters increase the computing time, but do no harm otherwise. We have the following theorem.

Theorem 5 *The original set of edits with linear inequalities involving only integer variables as THEN-conditions has a solution if and only if the dark shadow or a splinter resulting from the elimination of variable x_r has a solution.*

Theorem 5 follows immediately by noting that for arbitrary, fixed categorical values, it reduces to Theorem 3.

We now eliminate all integer-valued variables from the original set of inequality edits. During this process we may have to consider several different sets of edits and corresponding variables owing to the splinters that may arise. We consider each such set of edits (and corresponding variables) separately. If a set of edits contains a balance edit, which happens if this set of edits is a splinter, we apply the technique of Subsection 7.1 to eliminate that equality from this set. For a set of edits involving only inequality edits, we select a variable that has not yet been eliminated and proceed to eliminate this variable using the technique of this subsection. We continue until all integer variables in all sets of edits have been eliminated, and we are left with one or more sets of edits involving only categorical variables. The theory of Subsection 7.1 and a repeated application of Theorem 5 yields the following theorem.

Theorem 6 *A set of edits with THEN-conditions involving only integer variables has a solution if and only if any of the sets of edits involving only categorical variables arising after the elimination of all integer-valued variables has a solution.*

To check the existence of a solution to a set of edits involving only categorical variables one can use the methodology sketched in Section 5.

7.3 Error localisation: algorithm for categorical, continuous and integer data

After the preparations in the previous subsections, we are now able to state our algorithm for solving the error localisation problem for a mix of categorical, continuous and integer data. We denote the error localisation problem in categorical, continuous and integer data under consideration by P_I. To solve P_I we first apply the branch-and-bound algorithm presented in Section 5 without taking into account that some of the variables are integer-valued, *i.e.* we first treat the integer variables as being continuous. We denote the problem where integer variables are treated as continuous ones by P_C. P_C is the continuous error localisation problem.

Let c_{obj} denote the value of the objective function (13) for the best currently found solution to P_I, and S the set of currently best solutions to P_I. We initialise c_{obj} to ∞, and S to $\emptyset$. A solution to P_C not involving any integer variables is automatically also a solution to P_I. So, whenever we find a solution to P_C not involving any integer variables for which (13) is less than c_{obj}, we update c_{obj} with that value of (13) and set S equal to the current solution to P_C. Also, whenever we find a solution to P_C not involving any integer variables for which (13) is equal to c_{obj}, we add the current solution to P_C to S.

Whenever we find a solution to P_C involving integer variables for which (13) is at most equal to c_{obj}, we consider P_I. We test whether the variables involved in the current solution to P_C also constitute a solution to P_I. The basic idea of this test is illustrated in Figure 3 below.

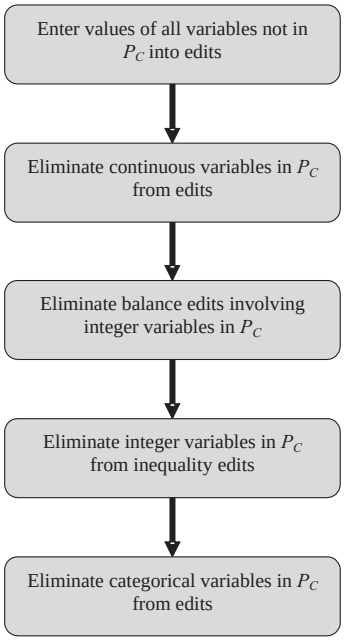

Figure 3: *The basic idea of the integrality test.*

For the integrality test we first fill in the values of the variables not involved in the current solution to P_C into the edits. Subsequently, we eliminate the continuous variables involved in the solution to P_C. This yields a system of edits (10) in which only the integer-valued and categorical variables involved in the solution to P_C occur.

Next, we eliminate all balance edits with integer-valued variables involved in the current solution to P_C in the manner described in Subsection 7.1. Subsequently, we eliminate all integer-valued variables involved in the current solution to P_C from all inequality edits in the manner described in Subsection 7.2. During this latter elimination process the original problem may be split into several subproblems owing to the splinters that may arise. Finally, we eliminate all categorical variables from each of these subproblems. For each subproblem we end up with a set of relations not involving any unknowns. Such a set of relations may be empty. If a set of relations we obtain in this way does not contain a self-contradicting relation, which is for instance (by definition) the case if the set of relations is empty, we have found a solution to P_I. In that case, if the value of (13) for the current solution to P_I is less than c_{obj} we update c_{obj} accordingly and set S equal to the current solution of P_I, else we add the current solution to P_I to S. If all sets of relations involving no unknowns contain a self-contradicting relation, none of the subproblems leads to a solution to P_I and the solution to P_C under consideration is not a solution to P_I. In that case c_{obj} is not updated, and we continue with finding solutions to P_C.

Note that in the above approach, the relatively time-consuming integrality test is only invoked once a solution to P_C with an objective value of c_{obj} or less involving integer-valued variables has been found, so generally only rather infrequently. We have the following theorem.

Theorem 7 *The above procedure finds all optimal solutions to P_I.*

Proof. We start by noting that Theorem 1, Subsection 7.1 and Theorem 6 show that if and only if any of the final sets of relations involving no unknowns obtained by eliminating all variables involved in a solution to P_C does not contain a self-contradicting relation, the original set of edits can be satisfied by modifying the values of the variables involved in this solution. Now, the branch-and-bound algorithm for categorical and continuous data can be used to find all solutions to P_C with an objective value (13) of c_{obj} or less, for any given value of c_{obj}. For each solution to P_C with a value for (13) equal to or less than c_{obj}, we test whether it is also a solution to P_I. The result of this test is conclusive. We update c_{obj} whenever we have found a better solution to P_I than the best one found so far. In other words, all potentially optimal solutions to P_I are considered by the procedure, and all optimal solutions to P_I are indeed identified as such. □

We illustrate the algorithm by means of a simple example involving only two integer-valued variables.

Example 2 *We consider a case with only two variables x_1 and x_2, and three edits given by*

$$-2x_2 + 5 \geq 0,$$

$$5x_1 - x_2 \geq 0, \tag{53}$$

$$-3x_1 + 2x_2 \geq 0.$$

Both variables are integer-valued, and their reliability weights equal one. The original, incorrect record is given by $x_1 = 1$, and $x_2 = 1$. We initialise c_{obj} to ∞, and S to $\emptyset$. We start by solving P_C. We select a variable, say x_1, and construct two branches: in the first branch we eliminate x_1 from the set of current edits, in the second branch we fix x_1 to its original value. If we eliminate x_1 from the set of current edits, we obtain (53) and $x_2 \geq 0$ as our new set of current edits. This new set of current edits is satisfied by the original value of x_2. Hence, we have found a solution to P_C, namely: change x_1. We test whether this is also a solution to P_I. To this end, we start by filling in the original value of x_2 into the original set of edits. We obtain the following set of edits involving only x_1.

$$5x_1 - 1 \geq 0, \tag{54}$$

$$-3x_1 + 2 \geq 0. \tag{55}$$

The dark shadow obtained by eliminating x_2 from (54) and (55) (see (50)) is given by

$$7 \geq 8,$$

which is clearly a self-contradicting relation. We therefore have to consider the splinters. In this simple case there are three splinters. For the first splinter we have to add the constraint

$$5x_1 = 1$$

to (54) and (55) (see (52)), for the second one the constraint

$$5x_1 = 2,$$

and for the third one the constraint

$$5x_1 = 3.$$

It is clear that none of these three splinters has an integer solution for x_1. This would also follow if we were to apply the proposed algorithm. We conclude that although changing x_1 is a solution to P_C, it is not a solution to P_I.

After this intermezzo during which we tested whether changing the value of only x_1 is a solution to $P_I$we continue with finding solutions to P_C. We now consider the branch where x_1 is fixed to its original value. The corresponding set of current edits is given by (53),

$$-x_2 + 5 \geq 0, \tag{56}$$

$$2x_2 - 3 \geq 0. \tag{57}$$

By eliminating x_2 we see that changing the value of only x_2 is a solution to P_C. We check whether this is also a solution to P_I. We fill in the original value of x_1 into the original set of edits. We obtain the system (53), (56) and (57). The dark shadow of (53) and (57) obtained by eliminating x_2 (see (50)) is given by

$$4 \geq 1,$$

and the dark shadow of (56) and (57) obtained by eliminating x_2 by

$$7 \geq 0.$$

The above relations are not self-contradicting, so we can conclude that changing the value of x_2 is a solution to P_I. We can even conclude that this is the only optimal solution to P_I. A feasible value for x_2 is 2.

The method described in this section may appear to be very slow in many cases. Indeed, it is not difficult to design a set of edits for which the method is extremely slow. However, we argue that in practice the situation is not so bad. First, like we already mentioned, the time-consuming algorithm to check potential solutions to P_I is only invoked once a new solution to P_C with an objective value less than or equal to the current value of c_{obj} involving integer variables has been found. In practice, the number of times that such a solution to P_C is found is in most cases rather limited.

Second, whenever we find a solution to P_C with an objective value less than or equal to the current value of c_{obj} we only have to test whether the variables involved in this particular solution also form a solution to P_I. Moreover, often one is only interested in solutions to the error localisation problem with a few variables, say 10 or less. We already argued in Section 5 that if a record requires more than, say, 10 values to be changed, it should not be edited automatically in our opinion as the statistical quality of the automatically edited record would be too low. This implies that the relatively time-consuming test described in this section involves only a few variables.

Third, the integrality test becomes only really time-consuming when many splinters

have to be considered. However, in most edits, either explicit or implicit ones, encountered in practice the coefficients of the integer variables equal –1 or +1. This is especially true for balance edits. For an integer variable with coefficient –1 or +1 the elimination from inequality edits will be exact, *i.e.* the dark shadow and the real shadow coincide and no splinters have to be generated. For balance edits involving integer variables with coefficients -1 or +1 no auxiliary variables have to be introduced in order to eliminate these edits. For such a balance edit the elimination can be performed very fast.

Finally, we can also resort to a heuristic approach based on the exact algorithm. In the next section such a heuristic procedure is described.

8 A heuristic procedure

At Statistics Netherlands we originally aimed to develop a software package for a mix of categorical and continuous data only. In order to achieve this aim a number of algorithms were considered. For an overview of the algorithms considered we refer to De Waal (2003a). Most of those algorithms have been implemented in prototype software, and have subsequently been evaluated. For an assessment of several algorithms on continuous data we refer to De Waal and Coutinho (2005). As a consequence of our work the algorithm described in Section 5 has been implemented in SLICE, our general software framework for automatic editing and imputation (*cf.* De Waal, 2001). Later the wish to extend the implemented algorithm to include integer-valued data arose. The algorithm described in Section 7 was developed to fulfil that wish. However, once this algorithm was developed we considered it to be too complex to implement and maintain in production software. We therefore decided not to implement the exact algorithm of Section 7, but instead to develop a simpler heuristic procedure based on the exact algorithm. That heuristic procedure, which is described below, has been implemented in version 1.5 of SLICE.

Only the integrality test for the integer-valued variables involved in a solution to P_C, *i.e.* a potential solution to P_I, differ for the exact algorithm and the heuristic procedure. In our heuristic procedure, we do not examine splinters; neither do we introduce auxiliary variables in order to eliminate balance edits.

Whenever we have to eliminate an integer-valued variable x_r from a pair of edits s and t in the heuristic checking procedure, we distinguish between two cases. If either edit s or edit t (or both) is a balance edit involving x_r, we examine whether the coefficient of x_r in the corresponding normalised THEN-condition (if both edits are balance edits, we examine both normalised THEN-conditions) equals +1 or -1. If this is not the case, we make the conservative assumption that no feasible integer value for x_r exists, and we reject the potential solution to the P_I. If both edits s and t are inequality edits, we eliminate x_r from these edits by determining the dark shadow (see (50)).

If several integer variables are involved in the solution to P_C under consideration, we repeatedly apply the above procedure until all these variables have been eliminated. If the resulting set of edits involving only categorical variables has a solution, the solution to P_C is also a solution to P_I (see Theorem 6). On the other hand, if the resulting set of edits does not have a solution we make the conservative assumption that the current solution to P_C is not a solution to P_I. This assumption is conservative as we do not check the splinters.

The above heuristic procedure is considerably easier to implement and maintain than the exact algorithm of Section 7. The price we have to pay for using the heuristic procedure instead of the exact algorithm of Section 7 is that we sometimes conclude that an integer solution does not exist, whereas in fact it does.

9 Computational results

In this section we provide computational results for the heuristic procedure described in Section 8. The exact algorithm described in Section 7 has not been implemented in a computer program, and has therefore not been evaluated. The experiments have been performed on a 1500 MHz PC with 256 MB of RAM. This PC was connected to a local area network.

The heuristic procedure has been tested on five realistic data sets. We have used realistic data sets rather than randomly generated synthetic data for our evaluation study, because we feel that the properties of realistic data are completely different than those of randomly generated data. Considering that a production version of SLICE for categorical and continuous data already existed, we decided to implement the heuristic procedure described in Section 8 directly in SLICE (version 1.5), without implementing it in prototype software first.

Our experiments have therefore been carried out by means of SLICE 1.5. This software package has been designed for use in the day-to-day routine at our statistical office. It has been optimised for robustness against mis-use and for ease of maintainability. It uses well-tested components that facilitate debugging. Moreover, the software stores different kinds of metadata, such as which fields are identified as being erroneous. SLICE 1.5 has not been optimised for speed. The speed of the software can definitely be improved upon. Compared to the prototype software for categorical and continuous data on which this production software is based, the production software is about 16 times or more slower (see De Waal and Quere, 2003, where similar data and edits were used as in the present article). The prototype software, however, could handle only a mix of categorical and continuous data, not integer-valued data.

SLICE 1.5 allows the user to specify several parameters, such as a maximum for the number of errors in a record, a maximum for the number of missing values in a record, the maximum computing time per record, the maximum number of (explicit

and implicit) edits in a node of the binary search tree, and a maximum for the number of determined solutions. In our evaluation experiments we did not set a limit for the number of missing values in a record. We have set the maximum number of (explicit and implicit) edits in a node to 3,000, and the maximum computing time per record to 60 seconds. In our experiments we have varied the maximum number of errors and the maximum for the number of determined solutions.

If a record cannot be made to satisfy all edits by changing at most the specified maximum number of errors, it is discarded by SLICE 1.5. A record is also discarded by SLICE 1.5 if it contains more missing values than the specified maximum. Whenever SLICE 1.5 has found N_{sol} solutions with the lowest value c_{low} for the objective function (13) found so far, where N_{sol} is the specified maximum number of determined solutions, it from then on searches only for solutions to the error localisation problem for which the value of the objective function (13) is strictly less than c_{low}. After SLICE 1.5 has solved the error localisation problem for a record, it returns at most N_{sol} solutions with the lowest value for the objective function (13). Owing to the use of the heuristic procedure of Section 8, these determined solutions may be suboptimal. In case the maximum number of edits in a node exceeds 3,000 or the maximum computing time per record exceeds 60 seconds, SLICE 1.5 returns the best solutions (if any) it has determined so far. So, even if the maximum number of edits in a node or the maximum computing time per record is exceeded, the heuristic procedure implemented in SLICE 1.5 may return a solution. For some records the heuristic procedure of SLICE 1.5 could not find a solution at all.

For Statistics Netherlands improving the efficiency of the data editing process for economic, and hence mainly numerical, data is much more important than for social, and hence mainly categorical, data. Therefore, the heuristic procedure has only been evaluated for purely numerical test data. In fact, all variables in the five data sets were integer-valued ones. The five evaluation data sets come from a wide range of business surveys. In Table 1 we give a brief description of each data set.

Table 1: *Description of the five evaluation data sets*

Name	Description
Data set A	structural business survey on enterprises in the photographic sector
Data set B	structural business survey on enterprises in the building and construction industry
Data set C	structural business survey on the retail sector
Data set D	survey on environmental expenditures
Data set E	Annual Business Inquiry

Data set E, the so-called Annual Business Inquiry data set, is one of the evaluation data sets from the EUREDIT project. The EUREDIT project (see http://www.cs.york.ac.uk/euredit) was a large international research and development project on statistical data editing and imputation involving 12 institutes, including

Statistics Netherlands, from seven different countries. The project lasted from March 2000 till March 2003. Important aims were the evaluation of current "in-use" methods for data editing and imputation, and the development and evaluation of a selected range of new or recent techniques for data editing and imputation. For more information on the methods examined in the EUREDIT project we refer to Chambers (2004). Owing to confidentiality reasons the branch of industry to which the businesses in data set E belong has not been made public.

To the best of our knowledge the five evaluation data sets are representative for many other data sets from business surveys. A good performance on the five evaluation data sets hence suggests that the performance on many other business survey data sets arising in practice will also be acceptable.

In Table 2 below we give a summary of the characteristics of the five evaluation data sets. In this table the number of integer-valued variables, the number of non-negativity constraints (*i.e.* constraints expressing that a variable should have a non-negative value), the number of inequality edits (excluding the non-negativity constraints), the number of balance edits, the total number of records, the number of inconsistent records (*i.e.* records failing edits or containing missing values), the total number of missing values, and the average number of errors per inconsistent record are listed.

Table 2: *Characteristics of the five evaluation data sets*

	Data set A	Data set B	Data set C	Data set D	Data set E
Number of integer variables	76	53	51	54	26
Number of non-negativity constraints	70	36	49	54	22
Number of inequality edits[a]	2	16	7	0	15
Number of balance edits	18	20	8	21	3
Total number of records	274	1,478	4,217	1,039	1,425
Number of inconsistent records	157	1,402	2,152	378	1,141
Total number of missing values	0	0	0	2,230	195
Average number of errors per inconsistent record	2.7	2.7	1.6	6.2	2.6

[a] Excluding non-negativity constraints

In all balance edits corresponding to the five evaluation data sets, the coefficients of the involved variables equal -1 or +1. Also, in all inequality edits corresponding to data sets A and C, the coefficients of the involved variables equal -1 or +1. In the equality edits corresponding to data sets B and E, however, many coefficients of the involved variables are not equal to -1 or +1.

We have compared the solutions determined by the heuristic procedure implemented in SLICE 1.5 to the optimal solutions. For purely numerical data, the edits (10) reduce to linear constraints, and the error localisation problem can easily be formulated as an integer programming problem (see, *e.g.*, Schaffer, 1987, and Riera-Ledesma and Salazar-González, 20033). We have therefore used a solver for integer programming

problems to determine the optimal solutions. For our evaluation study we have used CPLEX (*cf*. ILOG CPLEX 7.5 Reference Manual, 2001). Note that although the error localisation problem for numerical (either continuous or integer-valued) data can quite easily be solved by a solver for integer programming problems, the error localisation problem for a mix of numerical (either continuous or integer-valued) and categorical data quickly becomes very hard to solve for such a solver.

In Table 3 we give the number of records for which the heuristic procedure of SLICE 1.5, with the maximum number of errors set to 10, found an optimal solution, the number of records for which it could not find a (possibly suboptimal) solution at all, and the number of records for which it did find solution but exceeded the maximum computing time per record. In our evaluation study the maximum number of edits in a node was never exceeded. Note that records for which the heuristic procedure exceeded the maximum computing time may still be solved to optimality by this procedure.

Table 3: *Number of records that were optimally solved, could not be solved, and for which the maximum computing time per record was exceeded*

	Data set A	Data set B	Data set C	Data set D	Data set E
Number of optimally solved records	120	1,347	2,150	378	1039
Number of unsolved records	4	30	2	0	2
Number of records for which the maximum computing time (60 seconds) was exceeded	3	14	11	0	0

As described in Sections 7 and 8, the heuristic procedure of SLICE 1.5 consists of two parts: a branch-and-bound algorithm where all numerical variables are treated as being continuous ones and an integrality test. In order to assess the slow-down of the algorithm owing to the integrality test, we compare the computing times of the heuristic procedure to the computing times if all variables were continuous ones rather than integer-valued ones. The computing times if all variables were continuous ones are, for various maximum numbers of errors and maximum numbers of determined solutions, given in Table 4 below.

Table 4: *Computing times of the algorithm if the variables were continuous ones (in seconds)*

Parameters	Data set A	Data set B	Data set C	Data set D	Data set E
6 errors, 10 solutions	807	1,725	5,158	228	702
8 errors, 10 solutions	1,023	3,163	5,173	631	704
10 errors, 10 solutions	1,131	6,074	5,187	1,384	706
10 errors, 1 solution	1,088	5,626	5,065	1,348	651

The computational results of the heuristic procedure for various maximum numbers of errors and maximum numbers of determined solutions are given in Table 5 below. In this table we also give the increase in computing time (in per cents of the computing time for the corresponding case for continuous data) owing to the integrality test.

Table 5: *Computing times of the heuristic procedure in seconds (between brackets the increase in computing time owing to the integrality test in per cents of the computing time for continuous data)*

Parameters	Data set A	Data set B	Data set C	Data set D	Data set E
6 errors, 10 solutions	949 (18%)	1,877 (9%)	5,210 (1%)	238 (4%)	783 (12%)
8 errors, 10 solutions	1,056 (3%)	3,505 (11%)	5,223 (1%)	644 (2%)	962 (37%)
10 errors, 10 solutions	1,152 (2%)	6,959 (15%)	5,307 (2%)	1,397 (1%)	1,129 (60%)
10 errors, 1 solution	1,089 (0%)	6,652 (18%)	5,184 (2%)	1,348 (0%)	1,073 (65%)

For data sets A to D, the increase in computing time owing to the integrality test is rather small, namely between 0% and 18%. For data set E, however, the increase in computing time owing to the integrality test is quite large (up to 65%).

The effect of increasing the maximum number of errors on the relative computing time of the integrality test depends on the data set under consideration. For data sets A and D, the relative increase in computing time owing to the integrality test becomes less with increasing maximum numbers of errors. For data set B the relative increase in computing time owing to the integrality test gradually becomes more with increasing maximum numbers of errors. For data set C, this relative increase in computing time is more or less stable for different maximum numbers of errors. Finally, for data set E the relative increase in computing time grows rapidly with increasing maximum numbers of errors.

Determining several solutions instead of one leads to a limited increase in computing time. In Tables 4 and 5 we have given the computing times of SLICE 1.5, with the maximum number of errors set to 10, for the maximum number of determined solutions set to 10 and for the maximum number of determined solutions set to 1. In Table 5 the largest relative increase in computing time when determining at most 10 solutions instead of only one is for data set A. The computing time increases from 1,089 seconds to 1,152 seconds, an increase of approximately 6%. The largest relative increase in computing time owing to determining at most 10 solutions instead of only one is slightly higher in Table 4, namely approximately 8% for data sets B and D.

In Table 6 below we give the total number of erroneous fields according to the heuristic procedure of SLICE 1.5 and the exact algorithm implemented by means of CPLEX for the records that could be solved, possibly in a suboptimal manner, by means of the heuristic procedure.

The number of fields that are unnecessarily identified as being erroneous by the heuristic procedure was very small in our evaluation study. In other words, for the data

Table 6: *Total number of erroneous fields in solved records according to the heuristic procedure and the exact algorithm*

	Data set A	Data set B	Data set C	Data set D	Data set E
Exact algorithm for integer data (CPLEX)	378	3,424	3,526	2,362	2,919
Heuristic procedure (SLICE 1.5)	381	3,482	3,526	2,362	2,919

sets used in our evaluation study, the quality of the solutions determined by the heuristic procedure in terms of the total number of fields identified as erroneous is very good. In the worst case, data set B, the surplus of fields identified as being erroneous by the heuristic procedure in comparison to the number of fields identified as being erroneous by the exact algorithm implemented by means of CPLEX is less than 2% of the latter number of fields.

Finally, we examine the quality of the heuristic procedure in terms of the number of optimal solutions determined. We set both the maximum number of errors and the maximum number of solutions per record to 10. The reason for selecting the latter number is that for records with more than 10 optimal solutions to the error localisation problem, it is very hard to later select the correct solution, *i.e.* correctly identify the erroneous fields, anyway. For the records for which the heuristic procedure succeeded in determining an optimal solution, we compare the number of optimal solutions determined by the heuristic procedure to the number of optimal solutions determined by the exact algorithm implemented by means of CPLEX. The results are given in Table 7.

Table 7: *Number of optimal solutions of the heuristic procedure and the exact algorithm (between brackets the number of optimal solutions determined by the heuristic procedure in per cents of the number of optimal solutions determined by the exact algorithm)*

	Data set A	Data set B	Data set C	Data set D	Data set E
Exact algorithm for integer data (CPLEX)	701	6,609	11,404	474	6,207
Heuristic procedure (SLICE 1.5)	701(100%)	6,477(98%)	11,404 (100%)	474(100%)	4,828(78%)

For data sets A, C and D the heuristic procedure determined the same number of optimal solutions as the exact algorithm. Data sets B and E, the data sets for which the coefficients of the variables involved in the corresponding inequality edits often are unequal to -1 or +1, the number of optimal solutions determined by the heuristic procedure is less than the number of optimal solutions determined by the exact algorithm implemented by means of CPLEX. In particular, this is the case for data set E, where the number of optimal solutions determined by the heuristic procedure is only 78% of the number of optimal solutions determined by the exact algorithm. Data set E is

the only data set for which the number of inequality edits with coefficients unequal to -1 or +1 for the involved variables clearly outnumbers the number of balance edits, which probably explains our result. Note that despite the fact that the number of optimal solutions determined by the heuristic procedure for data set E is clearly less than the actual number of optimal solutions, the heuristic procedure does succeed in solving all records to optimality, except for two records for which it could not find a solution at all.

10 Discussion

In this article we have developed an exact algorithm for solving the error localisation problem for a mix of categorical, continuous and integer data. This algorithm is quite complex to implement and maintain in a software system, especially in a software system that is meant to be used routinely in practice. Based on this exact algorithm we have therefore also developed a much simpler heuristic procedure. This heuristic procedure has been implemented in our production software, SLICE 1.5. In this article we have also examined the performance of the heuristic procedure.

The exact algorithm and the heuristic procedure described in this article have a number of theoretical drawbacks. Both the exact algorithm and the heuristic procedure are extensions to an exact algorithm for continuous and categorical data (see Section 5). The computing time of this latter exact algorithm can, theoretically, be exponential in its input parameters, such as the number of variables, the number of edits and the maximum number of errors. For some data sets in our evaluation study, namely data sets B and D, this exponential increase in the computing time owing to an increase of the maximum number of errors is, unfortunately, also observed in practice. For some practical instances of the error localisation problem, this exponential increase in the computing time may be a problem. For such instances, one has to resort to other heuristic approaches, such as setting fields that are likely to be erroneous to "missing" in a pre-processing step (the exact algorithm for continuous and categorical data is generally faster for records with many missing values than for records with many erroneous values), or to an alternative algorithm altogether (see Section 1 and De Waal and Coutinho, 2005, for references to some papers on alternative approaches).

The computing time of the integrality test of the exact algorithm can, theoretically, also be exponential in the number of variables, the number of edits, and the maximum number of errors. In our evaluation study on the heuristic procedure, the increase in computing time owing to the integrality test is limited for most evaluation data sets. However, for data E the increase in computing time owing to the integrality test grows rapidly when increasing the maximum number of errors. Again, for some practical instances of the error localisation problem, this rapid increase in the computing time may be a problem, and one may have to resort to other approaches.

In principle, the number of erroneous fields identified by the heuristic procedure may

be (much) higher than the number of erroneous fields identified by an exact algorithm. In our evaluation study this has, however, not occurred. The number of fields identified as being erroneous by the heuristic procedure is for all evaluation data sets almost equal, and often even precisely equal, to the number of fields identified as being erroneous by an exact algorithm implemented by means of CPLEX.

Another potential drawback of the heuristic procedure is that the number of optimal solutions determined by this procedure can be (much) less than for an exact algorithm. In our evaluation study this has also not occurred. For most evaluation data sets, the number of optimal solutions determined by the heuristic procedure is equal or almost equal to the number of optimal solutions determined by an exact algorithm implemented by means of CPLEX. The only exception is data set E, where the number of optimal solutions determined by the heuristic procedure drops to about 78% of the number of optimal solutions determined by an exact algorithm implemented by means of CPLEX. Whereas the actual average number of optimal solutions is 5.4 (=6,207/1,039, see Tables 3 and 7) per optimally solved record if the maximum number of optimal solutions determined is set to 10, the heuristic procedure determines only 4.2 optimal solutions on the average. Fortunately, for our purposes at Statistics Netherlands this is an acceptable result.

As mentioned before, at Statistics Netherlands we aimed to implement an algorithm for a mix of categorical, continuous and integer data. Given the fact we had already implemented the algorithm for continuous and categorical data described in Section 5 in our production software, our main choice to be made was whether we would implement the exact algorithm described in Section 7 or the heuristic procedure of Section 8 in that production software. Considering the complexity of implementing and maintaining the exact algorithm in production software, we decided to implement the heuristic procedure instead the exact algorithm. Our, admittedly limited, experience with the heuristic procedure so far suggests that we have made a good choice here. For Statistics Netherlands, the benefits of using the heuristic procedure, in particular a considerable simplification in developing and maintaining the software in comparison to the exact algorithm of Section 7, outweigh the disadvantages, possibly worse and less solutions, of using the heuristic procedure instead of the exact algorithm. Despite the earlier mentioned theoretical drawbacks of the heuristic procedure, its computing speed and the quality of its solutions thus far appear to be fully acceptable for application in practice at Statistics Netherlands.

Acknowledgement

The author wishes to thank three anonymous referees for their useful comments on an earlier version of this article.

References

Barcaroli, G., C. Ceccarelli, O. Luzi, A. Manzari, E. Riccini and F. Silvestri (1995). The methodology of editing and imputation of qualitative variables implemented in SCIA. Internal Report, ISTAT, Rome.

Boskovitz, A., R. Goré and M. Hegland (2003). A logical formalisation of the Fellegi-Holt method of data cleaning. Report, Research School of Information Sciences and Engineering, Australian National University, Canberra.

Bruni, R., A. Reale, and R. Torelli (2001). Optimization techniques for edit validation and data imputation. *Proceedings of Statistics Canada Symposium 2001 "Achieving Data Quality in a Statistical Agency: a Methodological Perspective" XVIII-th International Symposium on Methodological Issues.*

Bruni, R. and A. Sassano (2001). Logic and optimization techniques for an error free data collecting. Report, University of Rome "La Sapienza".

Chambers, R. (2004). Methods Investigated in the EUREDIT Project. In: *Methods and Experimental Results from the EUREDIT Project*, J.R.H. Charlton (ed.) http://www.cs.york.ac.uk/euredit/.

Central Statistical Office (2000). Editing and calibration in survey processing. Report SMD-37, Ireland.

Chvátal, V. (1983). *Linear Programming*. W.H. Freeman and Company: New York.

Dantzig, G.B. and B. Curtis Eaves (1973). Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory (A) 14*, 288-297.

De Jong, A. (2002). Uni-Edit: standardized processing of structural business statistics in the Netherlands. UN/ECE Work Session on Statistical Data Editing, Helsinki.

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. In *Proceedings in Computational Statistics*, J.G. Bethlehem and P.G.M. Van der Heijden (eds.), New York: Physica-Verlag, 277-282.

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

De Waal, T. (2003*b*). Solving the error localization problem by means of vertex generation. *Survey Methodology*, 29, 71-79.

De Waal, T. and W. Coutinho (2005). Automatic editing for business surveys: an assessment for selected algorithms. *International Statistical Review*, forthcoming.

De Waal, T. and R. Quere (2003). A fast and simple algorithm for automatic editing of mixed data. *Journal of Official Statistics*, 19, 383-402.

Duffin, R.J. (1974). On Fourier's analysis of linear inequality systems. *Mathematical Programming Studies*, 1, 71-95.

Fellegi, I.P. and D. Holt (1976). A systematic approach to automatic edit and imputation. *Journal of the American Statistical Association*, 71, 17-35.

Garfinkel, R.S., A.S. Kunnathur and G.E. Liepins (1986). Optimal imputation of erroneous data: categorical data, general edits. *Operations Research*, 34, 744-751.

Garfinkel, R.S., A.S. Kunnathur and G.E. Liepins (1988). Error localization for erroneous data: continuous data, linear constraints. *SIAM Journal on Scientific and Statistical Computing*, 9, 922-931.

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*, Cox, Binder, Chinnappa, Christianson & Kott (eds.), John Wiley & Sons, Inc., 385-401.

Granquist, L. (1997). The new view on editing. *International Statistical Review*, 65, 381-387.

Granquist, L. and J. Kovar (1997). Editing of survey data: how much is enough?. In *Survey Measurement*

and Process Quality, Lyberg, Biemer, Collins, De Leeuw, Dippo, Schwartz & Trewin (eds.), John Wiley & 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. (2002). Selective editing by means of plausibility indicators. UN/ECE Work Session on Statistical Data Editing, Helsinki.

Hoogland, J. and E. Van der Pijll (2003). Evaluation of automatic versus manual editing of production statistics 2000 trade & transport. UN/ECE Work Session on Statistical Data Editing, Madrid.

ILOG CPLEX 7.5 Reference Manual (2001). ILOG, France.

Kalton, G. and D. Kasprzyk (1986). The treatment of missing survey data. *Survey Methodology*, 12, 1-16.

Kovar, J. and P. Whitridge (1990). Generalized edit and imputation system: overview and applications. *Revista Brasileira de Estadistica*, 51, 85-100.

Kovar, J. and P. Whitridge (1995). Imputation of business survey data. In *Business Survey Methods*, Cox, Binder, Chinnappa, Christianson & Kott (eds.), New York: John Wiley & Sons, Inc., 403-423.

Liepins, G.E., R.S. Garfinkel and A.S. Kunnathur (1982). Error localization for erroneous data: A survey. *TIMS/Studies in the Management Sciences*, 19, 205-219.

McKeown, P.G. (1984). A mathematical programming approach to editing of continuous survey data. *SIAM Journal on Scientific and Statistical Computing*, 5, 784-797.

Nemhauser, G.L. and L.A. Wolsey (1988). *Integer and Combinatorial Optimisation.* John Wiley & Sons, New York.

Pannekoek, J. and T. De Waal (2005). Automatic editing and imputation for business surveys: the dutch contribution to the EUREDIT project. *Journal of Official Statistics*, forthcoming.

Pugh, W. (1992). The Omega test: a fast and practical integer programming algorithm for data dependence analysis. *Communications of the ACM 35*, 102-114.

Pugh, W. and D. Wonnacott (1994). Experiences with constraint-based array dependence analysis. In *Principles and Practice of Constraint Programming, Second International Workshop*. *Lecture Notes in Computer Science 768*, Berlin: Springer-Verlag.

Ragsdale, C.T. and P.G. McKeown (1996). On solving the continuous data editing problem. *Computers & Operations Research*, 23, 263-273.

Riera-Ledesma, J. and J.J. Salazar-González (2003). New algorithms for the editing and imputation problem. UN/ECE Work Session on Statistical Data Editing, Madrid.

Sande, G. (1978). An algorithm for the fields to impute problems of numerical and coded data. Technical report, Statistics Canada.

Schaffer, J. (1987). Procedure for solving the data-editing problem with both continuous and discrete data types. *Naval Research Logistics*, 34, 879-890.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. New York: John Wiley & Sons.

Stoop, J.R. (2003). The best piece of CherryPie (in Dutch). Internal report (BPA number: 2098-03-TMO), Voorburg: Statistics Netherlands.

Todaro, T.A. (1999). Overview and evaluation of the AGGIES automated edit and imputation system. UN/ECE Work Session on Statistical Data Editing, Rome.

Williams, H.P. (1976). Fourier-Motzkin elimination extension to integer programming. *Journal of Combinatorial Theory (A)*, 21, 118-123.

Williams, H.P. (1986). Fourier's method of linear programming and its dual. *American Mathematical Monthly*, 93, 681-695.

Winkler, W.E. (1996). State of statistical data editing and current research problems. UN/ECE Work Session on Statistical Data Editing, Rome.

Winkler, W.E. (1998). Set-covering and editing discrete data. Statistical Research Division Report 98/01, US Bureau of the Census, Washington, D.C.

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

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