Statistics Netherlands

Discussion Paper

Error localization as a mixed integer problem with the editrules package

The views expressed in this paper are those of the author and do not necessarily reflect the policies of Statistics Netherlands.

2014 | 07

Edwin de Jonge
Mark van der Loo

Error localization is the problem of finding out which fields in raw data records contain erroneous values. The `editrules` extension package for the `R` environment for statistical computing was recently extended with a module that allows for error localization based on a mixed integer programming formulation (MIP). In this paper we describe the MIP formulation of the error localization problem for the case of numerical, categorical, or mixed numerical and categorical datasets. We introduce a MIP formulation that is a generalization of both linear as well as categorical restrictions. We discuss the numerical boundaries within which a MIP solver generates a stable solution and give directions on changing them to your own needs. The new module is benchmarked against a previously available module, which is based on a branch-and-bound approach. The benchmark shows that the MIP-based approach is significantly faster. Trade-offs between the branch-and-bound and MIP approaches are discussed as well.

Contents

1 Introduction

Analyses of data are often hindered by occurrences of incomplete or inconsistent raw data records. The process of locating and correcting such errors is referred to as *data editing*, and it has been estimated that National Statistics Institutes may spend up to 40% of their resources on this process (De Waal et al., 2011). Moreover, data often must obey many cross-variable consistency rules which significantly complicate the data editing process. Indeed, Winkler (1999) mentions household surveys where records have to obey 250, 300 or even 750 user-defined interrelated consistency rules. For these reasons, considerable attention is paid to the development of data editing methods that can be automated.

1.1 Error localization

Automated as well as manual data editing strategies for a data record typically consist of three steps:

1. Find out which consistency rules a record violates;
2. Find out which fields in a record cause those violations;
3. Replace the values in those fields with better estimates, such that no rule is violated.

The second step is usually referred to as the *error localization* problem, which is the focus of the current paper. Although it is widely recognized that data editing is a necessary step in the statistical process, the amount of changes made to the data should obviously be minimized to avoid introducing bias in estimations based on the edited data. This then leads to the following minimization problem.

> Given a record of n variables, subject to a number of possibly multivariate consistency rules. Find the smallest (weighted) subset of fields, such that after replacement of their values, the record violates no rules.

This minimization problem is named after Fellegi and Holt (1976), who first formulated and solved the problem for the case of categorical data. Error localization has been extensively discussed in literature[*)], so we will suffice with a few remarks. First, the search space related to the minimization problem grows exponentially with the number of fields, rendering a brute-force approach that runs through all possible solution candidates computationally unfeasible. Second, the problem is complicated by the occurrence of *implied rules*. That is, the solution set must not only allow the record to obey the original, user-defined set of rules, but also rules that are logically or arithmetically implied by the original set. To cope with these complications several algorithmic approaches have been developed, two of which are worth mentioning in this context. The first is the branch-and-bound approach developed by De Waal and Quere (2003). The second is an approach based on mixed-integer programming (MIP) described in De Waal et al. (2011).

[*)] See De Waal et al. (2011) and references therein.

1.2 The editrules package

Over the past decade the R statistical environment has received a surge in popularity. As a consequence it has been extended with many user-built packages that allow for statistical analyses of data. However, the number of packages specifically aimed at data editing seems to be somewhat limited, except possibly in the area of imputation. The R package editrules (De Jonge and van der Loo, 2013) was developed to help to bridge the gap between raw data retrieval and data analysis with R. The main purpose of the package is to provide a convenient interface to define data consistency rules (often referred to as *edit rules*) in R and to confront them with data. Furthermore, the package allows for basic rule manipulation (deriving new rules, finding inconsistencies, *etc.*) and for error localization functionality. As such, the package does not offer functionality to correct data. Rather, it is aimed at identifying the set of solutions to an error localization problem: the second step mentioned in the data editing strategy above. Previous developments of the package have been described in De Jonge and Van der Loo (2011); Van der Loo and De Jonge (2011) and Van der Loo et al. (2011).

The `editrules` package offers a toolbox that allows users to work with numerical, categorical or mixed-type data editing rules. Up until now, error localization was performed by an implementation of the branch-and-bound algorithm described by De Waal (2003). The main disadvantage of this approach is that the branch-and-bound algorithm has $\mathcal{O}(2^n)$ worst-case time and memory complexity, where n is the number of variables occurring in a connected set of rules. Moreover, the branch-and-bound solver is written in pure R, making it intrinsically slower than a compiled language implementation. The main advantages of this approach are the ease of implementation and the opportunity for users to exert fine-grained control over the algorithm.

As stated before, the error localization problem can be translated to a mixed-integer programming problem. This allows us to reuse well-established results from the field of linear and mixed-integer programming. Indeed, many advanced algorithms for solving such problems have been developed, and in many cases implementations in a compiled language are available under a permissive license. In `editrules`, the solver of the `lp_solve` library (Berkelaar et al., 2010) is used through R's `lpSolveAPI` package (Konis, 2011). The `lp_solve` library is written in `ANSI C` and has been tried and tested extensively.

The strategy to solve error localization problems through this library from R therefore consists of translating the problem to a suitable mixed-integer programming problem, feeding this problem to `lpSolveAPI`, and translating the results back to an error location. It is necessary to distinguish between:

- linear restrictions on purely numerical data,
- restrictions on purely categorical data, and
- conditional restrictions on mixed-type data,

since restriction for each data type calls for a different translation to a MIP problem.

The second part of this paper will focus on how to translate these types of error localization problems to a mixed-integer formulation, paying attention to both theoretical and practical details. In Section 3 attention is payed to numerical stability issues, Section 4 is devoted to examples in R code and Section 5 describes benchmark results. Conclusions and a further outlook are described in Section 6.

2 Error localization and mixed integer programming

A mixed integer programming problem is an optimization problem that can be written in the form

$$\begin{aligned} &\text{Minimize } f(\boldsymbol{z}) = \boldsymbol{c}^T\boldsymbol{z}; \\ &\text{s.t. } \boldsymbol{R}\boldsymbol{z} \leq \boldsymbol{d}, \end{aligned} \tag{1}$$

where $\boldsymbol{c}$ is a constant vector and $\boldsymbol{z}$ is a vector consisting of real and integer coefficients. One usually refers to $\boldsymbol{z}$ as the *decision vector* and the inner product $\boldsymbol{c}^T\boldsymbol{z}$ as the *objective function*. Furthermore, $\boldsymbol{R}$ is a coefficient matrix and $\boldsymbol{d}$ a vector of upper bounds. Formally, the elements of $\boldsymbol{c}$, $\boldsymbol{R}$ and $\boldsymbol{d}$ are limited to the rational numbers (Schrijver, 1998). This is never a problem in practice since we are always working with a computer representation of numbers.

The name *mixed-integer programming* stems from the fact that $\boldsymbol{z}$ contains continuous as well as integer variables. When $\boldsymbol{z}$ consists solely of continuous or integer variables, Problem (1) reduces respectively to a *linear* or an *integer programming* problem. An important special case occurs when the integer coefficients of $\boldsymbol{z}$ may only take values from {0,1}. Such variables are often called binary variables. It occurs as a special case since defining $\boldsymbol{z}$ to be integer and applying the appropriate upper bounds yields the same problem.

Mixed integer programming is well understood and several software packages are available that implement efficient solvers. Most MIP software support a broader, but equivalent, formulation of the MIP problem, allowing the set of restrictions to include inequalities as well as equalities. As a side note we mention that under equality restrictions, solutions for the integer part of $\boldsymbol{z}$ are only guaranteed to exist when the equality restrictions pertaining to the integer part of $\boldsymbol{z}$ are *totally unimodular*[†)]. However, as we will see below, restrictions on $\boldsymbol{z}$ are always inequalities in our case, so this is of no particular concern to us.

In this paper we reformulate Fellegi Holt error localization (Fellegi and Holt, 1976) for numerical, categorical and mixed-type restrictions in terms of MIP problems. The precise reformulations of the error localization problem for the three types of rules are different, but in each case the objective function is of the form

$$\boldsymbol{w}^T\boldsymbol{\Delta}, \tag{2}$$

where $\boldsymbol{w}$ is a vector of positive weights and $\boldsymbol{\Delta}$ a vector of binary variables, one for each variable in the original record, that indicates whether its value should be replaced. More precisely, for a record $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)$ of n variables, we define

$$\Delta_i = \begin{cases} 1 \text{ if the value of } r_i \text{ must be replaced} \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$

This objective function obviously meets the requirement that the minimal (weighted) number of variables should be replaced. In general, a record may contain numeric, categorical or both types of data and restrictions may pertain to either one or both data types. To distinguish between the data types below we shall write $\boldsymbol{r} = (\boldsymbol{v}, \boldsymbol{x})$ where $\boldsymbol{v}$ represents the categorical and $\boldsymbol{x}$ the numerical part of $\boldsymbol{r}$.

[†)] This means that every square submatrix of the coefficient matrix $\boldsymbol{R}$ pertaining to the integer part of $\boldsymbol{z}$ has determinant 0 or ± 1.

For an error localization problem, the restrictions of Problem (1) consist of two parts, which we denote

$$\begin{bmatrix} \boldsymbol{R}^H \\ \boldsymbol{R}^0 \end{bmatrix} \boldsymbol{z} \leq \begin{bmatrix} \boldsymbol{d}^H \\ \boldsymbol{d}^0 \end{bmatrix}. \tag{4}$$

Here, the restrictions indicated with H represent a matrix representation of the user-defined (hard) restrictions that the original record $\boldsymbol{r}$ must obey. The vector $\boldsymbol{z}$ contains at least a numerical representation of the values in a record $\boldsymbol{r}$ and the binary variables $\boldsymbol{\Delta}$. An algorithmic MIP-solver will iteratively alter the values of $\boldsymbol{z}$ until a solution satisfying (4) is reached. To make sure that the objective function reflects the (weighted) number of variables altered in the process, the restrictions in $\boldsymbol{R}^0$ serve to make sure that the values in $\boldsymbol{z}$ that represent values in $\boldsymbol{r}$ cannot be altered without setting the corresponding value in $\boldsymbol{\Delta}$ to 1.

Summarizing, in order to translate the error localization problem for the special cases of linear, categorical or conditional mixed-type restrictions to a general mixed integer problem, for each case we need to properly define $\boldsymbol{z}$, the restriction set $\boldsymbol{R}^H\boldsymbol{z} \leq \boldsymbol{d}^H$ and the restriction set $\boldsymbol{R}^0\boldsymbol{z} \leq \boldsymbol{d}^0$.

2.1 Linear restrictions

For a numerical record $\boldsymbol{x}$ taking values in $\mathbb{R}^n$, a set of linear restrictions can be written as

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \tag{5}$$

where in `editrules`, we allow the set of restrictions to contain equalities, inequalities ($\leq$) and strict inequalities ($<$). The formulation of these edit rules is very close to the formulation of the original MIP problem of Eq. (1). The vector to minimize over is defined as follows:

$$\boldsymbol{z} = (x_1, x_2, \ldots, x_n, \Delta_1, \Delta_2, \ldots, \Delta_n). \tag{6}$$

with the Δ_i as in Eq. (3). The set of restrictions $\boldsymbol{R}^H\boldsymbol{z} \leq \boldsymbol{d}^H$ is equal to the set of restrictions of Eq. (5), except in the case of strict inequalities. The reason is that while `editrules` allows the user to define strict inequalities ($<$), the `lpsolve` library used by `editrules` only allows for inclusive inequalities ($\leq$). For this reason, strict inequalities of the form $\boldsymbol{a}^T\boldsymbol{x} < b$ are rewritten as $\boldsymbol{a}^T\boldsymbol{x} \leq b - \epsilon$, with ϵ a suitably small positive constant.

In the case of linear edits, the set of constraints $\boldsymbol{R}^0\boldsymbol{z} \leq \boldsymbol{d}^0$ consists of pairs of the form

$$\begin{aligned} x_i - M\Delta_i &\leq x_i^0 \\ -x_i - M\Delta_i &\leq -x_i^0 \end{aligned} \tag{7}$$

for $i = 1, 2, \ldots, n$. Here The x_i^0 are the actual observed values in the record and M is a suitably large positive constant allowing x_i to vary between $x_i^0 - M$ and $x_i^0 + M$. It is not difficult to see that if x_i is different from x_i^0 then Δ_i must equal 1. For, if we choose $\Delta_i = 0$ we obtain the set of restrictions

$$x_i^0 \leq x_i \leq x_i^0 \tag{8}$$

which states that x_i equals x_i^0

Example 1. *Consider a record with business survey data, consisting of the variables* Number of staff p *and* Personnel cost c. *We have the rules* $p \geq 0$, $c \geq 0$ *and* $c \geq p$. *The latter rule expresses the notion that for each staff member, more than one monetary unit is spent. Given two observed values* p^0 *and* c^0, *disobeying one or more of the rules, the MIP problem for error localization has the following form.*

$$\underset{(\boldsymbol{x},\boldsymbol{\Delta})\in\mathbb{R}^2\times\{0,1\}^2}{\text{Minimize}} \ \Delta_p + \Delta_c$$

$$s.t. \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & -M & 0 \\ -1 & 0 & -M & 0 \\ 0 & 1 & 0 & -M \\ -1 & 1 & 0 & -M \end{bmatrix} \begin{bmatrix} p \\ c \\ \Delta_p \\ \Delta_c \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ p^0 \\ -p^0 \\ c^0 \\ -c^0 \end{bmatrix}.$$

Here, the first three rows in the set of restrictions represent the consistency rules while the other rows connect the indicator variables $\boldsymbol{\Delta} = (\Delta_p, \Delta_c)$ *with* p *and* c. □

2.2 Categorical restrictions

Categorical records $\boldsymbol{v} \in D$ take values in a Cartesian product domain

$$D = D_1 \times D_2 \times \cdots \times D_m, \tag{9}$$

where each D_i is a finite set of categories for the i^{th} categorical variable. The category names are unimportant so we write

$$D_i = \{1, 2, \ldots, |D_i|\}. \tag{10}$$

The total number of possible value combinations $|D|$ is equal to the product of the $|D_i|$.

A categorical edit is a subset F of D where records are considered invalid, and we may write

$$F = F_1 \times F_2 \times \cdots \times F_m, \tag{11}$$

where each F_i is a subset of D_i. It is understood that if a record $\boldsymbol{v} \in F$ then the record violates the edit. Hence, categorical edits are negatively formulated (they specify the region of D where $\boldsymbol{v}$ may not be) in contrast to linear edits which are positively formulated (they specify the region of $\mathbb{R}^n$ where $\boldsymbol{x}$ must be). To be able to translate categorical edits to a MIP problem, we need to specify $\overline{F}$, such that if $\boldsymbol{v} \in \overline{F}$ then $\boldsymbol{v}$ satisfies e. Here, $\overline{F}$ is the complement of F in D, which can be written as

$$\begin{aligned} \overline{F} = {} & \overline{F}_1 \times D_2 \times \cdots \times D_m \\ \cup \quad & D_1 \times \overline{F}_2 \times \cdots \times D_m \cup \cdots \cup D_1 \times D_2 \times \cdots \times \overline{F}_m, \end{aligned} \tag{12}$$

where for each variable v_i, $\overline{F}_i$ is the complement of F_i in D_i. Observe that Eq. (12) states that if at least one $v_i \in \overline{F}_i$, then $\boldsymbol{v}$ satisfies e. Below, we will use this property and construct a linear relation that counts the number of $v_i \in \overline{F}_i$ over all variables.

To be able to formulate the Fellegi Holt-problem in terms of a MIP problem, we first associate with each categorical variable v_i a binary vector $\boldsymbol{d}$ of which the coefficients are defined as follows (see also Eq. (9)).

$$d_\lambda(v_i) = \begin{cases} 1 \text{ if } v_i = \lambda \\ 0 \text{ otherwise,} \end{cases} \tag{13}$$

where $\lambda \in D_i$. Thus, each element of $\boldsymbol{d}(v_i)$ corresponds to one category in D_i. It is zero everywhere except at the value of $v_i \in D_i$. We will write $\boldsymbol{d}(\boldsymbol{v})$ to indicate the concatenated vector $(\boldsymbol{d}(v_1), \dots, \boldsymbol{d}(v_m))$ which represents a complete record. Similarly, each edit can be represented by a binary vector $\boldsymbol{e}$ given by

$$\boldsymbol{e} = \left(\bigvee_{\lambda \in \overline{F}_1} \boldsymbol{d}(\lambda), \dots, \bigvee_{\lambda \in \overline{F}_m} \boldsymbol{d}(\lambda) \right), \tag{14}$$

where we interpret 1 and 0 as TRUE and FALSE respectively and the logical 'or' ($\vee$) is applied element-wise to the coefficients of $\boldsymbol{d}$. The above relation can be interpreted as stating that $\boldsymbol{e}$ represents the valid value combinations of variables contained in the edit.

To set up the hard restriction matrix $\boldsymbol{R}^H$ of Eq. (4), we first impose the obvious restriction that each variable can take but a single value:

$$\sum_{\lambda \in D_i} d_\lambda(v_i) = 1, \tag{15}$$

for $i = 1, 2, \dots, m$. It is now not difficult to see that the demand (Eq. (12)) that at least one of the $v_i \in \overline{F}_i$ may be written as

$$\boldsymbol{e}^T \boldsymbol{d}(\boldsymbol{v}) \geq 1. \tag{16}$$

Equations (15) and (16) constitute the hard restrictions, stored in $\boldsymbol{R}^H$.

Using the binary vector notation for $\boldsymbol{v}$, and adding the Δ-variables that indicate variable change, the vector to minimize over (Eq. (1)) is written as

$$\boldsymbol{z} = (\boldsymbol{d}(\boldsymbol{v}), \Delta_1, \Delta_2, \dots, \Delta_m). \tag{17}$$

To ensure that a change in v_i results in a change in Δ_i, the matrix $\boldsymbol{R}^0$ contains the restrictions

$$d_{\lambda^0}(v_i) = 1 - \Delta_i, \tag{18}$$

for $i = 1, 2, \dots, m$. Here, $\lambda^0 \in D_i$ is the observed value for variable v_i. One may check, using Eq. (13), that the above equation can only hold when either $v_i = \lambda^0$ and $\Delta_i = 0$ (the original value is retained) or $v_i \neq \lambda^0$ and $\Delta_i = 1$ (the value changes).

Example 2. *Consider a two-variable record from the census with the variables* Marital status m *and* Age class a. *We have* $\boldsymbol{v} = (m, a) \in D$ *where*

$$D = D_m \times D_a = \{married, unmarried\} \times \{child, adult\}.$$

Using the binary representation we see that a married adult is represented by the vector $\boldsymbol{v}^0 = (\boldsymbol{d}(married), \boldsymbol{d}(adult)) = (1, 0, 0, 1)$. *The rule that states* ``A child cannot be married'' *translates to*

$$F = F_m \times F_a = \{married\} \times \{child\}$$

which gives $\overline{F}_m = \{unmarried\}$ *and* $\overline{F}_a = \{adult\}$. *Using Eq.* (14) *we get* $\boldsymbol{e} = (0, 1, 0, 1)$ *and one may verify that* $\boldsymbol{e}^T \boldsymbol{d}(married, child) = 0$ *and therefore invalid (see 16). For* $\boldsymbol{v}^0$, *the MIP problem*

for error localiation now looks like this.

$$\underset{(\boldsymbol{v},\boldsymbol{\Delta})\in D\times\{0,1\}^2}{\text{Minimize}}\ \Delta_m + \Delta_a \tag{19}$$

$$\text{s.t.}\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}\begin{bmatrix} d_{married}(m) \\ d_{unmarried}(m) \\ d_{child}(a) \\ d_{adult}(a) \\ \Delta_m \\ \Delta_a \end{bmatrix}\begin{matrix} \geq \\ = \\ = \\ = \\ = \end{matrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Here, the first row represents the edit rule, the second and third force that each variable can take but one value (Eq. 16), and the last two rows connect the indicator variables Δ_m and Δ_a with the value of m and a (Eq. 18). □

2.3 Mixed-type restrictions

Records $\boldsymbol{r}$ containing both numerical and categorical data can be denoted as a concatenation of categorical and numerical variables taking values in $D \times \mathbb{R}^n$:

$$\boldsymbol{r} = (v_1, \ldots, v_m, x_1, \ldots, x_n) = (\boldsymbol{v}, \boldsymbol{x}), \tag{20}$$

where D is defined in Eq. (9). As stated above, categorical edits are usually defined negatively as a region of D that is disallowed while linear edits define regions in $\mathbb{R}^n$ that are allowed. We may choose a negative formulation of edits containing both variable types by defining a single edit E as follows:

$$E = \{\boldsymbol{r} \in D \times \mathbb{R}^n : \boldsymbol{v} \in F \wedge \boldsymbol{x} \in P\}, \tag{21}$$

where $F \subseteq D$ and P is a convex subset of $\mathbb{R}^n$ defined by a (possibly empty) set of k linear inequalities of the form $\boldsymbol{a}^T\boldsymbol{x} > b$. It is understood that if $\boldsymbol{r} \in E$, then $\boldsymbol{r}$ violates the edit. An example of a restriction pertaining to a categorical and a numerical variable is `A company employs staff if and only if it has positive personell cost', The corresponding edit can be denoted as $\{\text{no staff}\} \times \{c > 0\}$.

To obtain a positive reformulation, we first negate the set membership condition and apply basic rules of proposition logic:

$$\begin{aligned} & \neg\left(\boldsymbol{v} \in F \wedge \boldsymbol{x} \in P\right) \\ \Leftrightarrow\ & \neg\left(\boldsymbol{v} \in F \wedge \boldsymbol{a}_1^T\boldsymbol{x} > b_1 \wedge \cdots \wedge \boldsymbol{a}_k^T\boldsymbol{x} > b_k\right) \\ \Leftrightarrow\ & \boldsymbol{v} \in \overline{F} \vee \boldsymbol{a}_1^T\boldsymbol{x} \leq b_1 \vee \cdots \vee \boldsymbol{a}_k^T\boldsymbol{x} \leq b_k. \end{aligned} \tag{22}$$

This then yields a positive formulation of E. That is, a record $\boldsymbol{r}$ satisfies E if and only if

$$\boldsymbol{r} \in \overline{E} \Leftrightarrow \bigvee_{i=1}^{m} v_i \in \overline{F}_i \vee \bigvee_{j=1}^{k} \boldsymbol{a}_j^T\boldsymbol{x} \leq b_j. \tag{23}$$

Observe that this formulation allows one to define multiple disconnected regions in $D \times \mathbb{R}^n$ containing valid records using just a single edit. For example, one may define a numeric variable to be either smaller than 0 or larger than 1. This type of restriction cannot be formulated using just linear numerical restrictions.

This formulation is both a generalization of linear inequality (Eq. 5) and categorical edits (Eq. 11). Choosing $k = 0$, we get $P = \mathbb{R}^n$ and only the categorical part remains. Similarly, choosing $F = \emptyset$, only the disjunction of linear inequalities remains. A system of linear equations that must simultaneously be obeyed like in Eq. (5) can be obtained by defining multiple edits E, each containing a single linear restriction.

The definition in Eq. (22) can be rewritten as a `conditional edit' by using the implication replacement rule from propositional logic which states that $\neg p \vee q$ may be replaced by $p \Rightarrow q$. If we limit Eq. (22) to a single inequality, we obtain the normal form of De Waal (2003).

$$\boldsymbol{v} \in F \Rightarrow \boldsymbol{a}^T\boldsymbol{x} \leq b. \tag{24}$$

If we choose $F = \emptyset$ and leave two inequalities we obtain a conditional edit on numerical data:

$$\boldsymbol{a}_1^T\boldsymbol{x} > b_1 \Rightarrow \boldsymbol{a}_2^T\boldsymbol{x} \leq b_2. \tag{25}$$

Writing mixed-type edits in conditional form seems more user-friendly as they can directly be translated into an `if` statement in a scripting language. Finally, note that equalities can be introduced by defining pairs of edits like so:

$$\begin{cases} \boldsymbol{v} \in F \Rightarrow \boldsymbol{a}^T\boldsymbol{x} \leq b \\ \boldsymbol{v} \in F \Rightarrow -\boldsymbol{a}^T\boldsymbol{x} \leq -b. \end{cases} \tag{26}$$

To reformulate Eq. (22) as a MIP problem, we first define binary variables ℓ_j that indicate whether $\boldsymbol{x}$ obeys $\boldsymbol{a}_k^T\boldsymbol{x} > b$:

$$\ell_j = \begin{cases} 0 \text{ when } \boldsymbol{a}_j^T\boldsymbol{x} \leq b_j \\ 1 \text{ when } \boldsymbol{a}_j^T\boldsymbol{x} > b_j \end{cases} \tag{27}$$

Using the or-form of the set condition (Eq. (22)) we can write the mixed-data edit as

$$\boldsymbol{e}^T\boldsymbol{d}(\boldsymbol{v}) + \sum_{j=1}^{k}(1 - \ell_j) \geq 1. \tag{28}$$

Recall from Eqs. (14) and (13) that $\boldsymbol{e}$ is the binary vector representation of a categorical edit and $\boldsymbol{d}(\boldsymbol{v})$ the binary vector representation of a categorical record. In the above equation, the `+' is the arithmetic translation of the logical `∨' operator in Eq. (22) that connects the categorical with the linear restrictions. When any of the two terms is positive, record r satisfies edit E.

Rules of this form constitute the user-defined part of the $\boldsymbol{R}^H$ part of the restriction matrix. To explicitly identify ℓ_j with the linear restrictions we also add

$$\boldsymbol{a}_j^T\boldsymbol{x} \leq b_j + M\ell_j, \tag{29}$$

to $\boldsymbol{R}^H$ with M a suitably large positive constant. Indeed, if $\ell_j = 0$, the inequality $\boldsymbol{a}_j^T\boldsymbol{x} \leq b_j$ is enforced and Eq. 28 always is satisfied. When $\ell_j = 1$ the whole restriction can hold regardless of whether the inequality holds. Finally, similar to the purely categorical case we need to add restrictions on the binary representation of $\boldsymbol{v}$ as in Eq. (15), so Eq. (15), Eq. (28) and Eq. (29) constitute $\boldsymbol{R}^H$.

There may be multiple mixed-type edits, each yielding one or more l indicator variables for each edit. The decision vector for the MIP problem may therefore be written as

$$\boldsymbol{z} = (\boldsymbol{d}(\boldsymbol{v}), \boldsymbol{x}, \Delta_1, \dots, \Delta_m, \dots, \Delta_{m+n}, \ell_1, \dots, \ell_K), \tag{30}$$

where K is the total number of linear edits occurring in all the mixed-type edits. Finally, the $\boldsymbol{R}^0$ matrix connecting the change indicator variables (Δ) with the actual recorded values consists of

the union of the restrictions for categorical variables (Eq. (18)) and those for numerical variables (Eq. (7)).

Example 3. *We consider a record* $\boldsymbol{r}$ *with the variables* type of business t*, which takes values in* $D_t = \{sp, other\}$*, where ``sp'' stands for ``sole proprietorship'',* personnel cost $c \in \mathbb{R}$ *and* number of staff $p \in \mathbb{R}$*. Hence, we have* $\boldsymbol{r} = (t, p, c) \in D_t \times \mathbb{R}^2$*. We impose the following rules on* $\boldsymbol{r}$*:* $p \geq 0$*,* $c \geq 0$*,* $c \geq p$ *and if the* business type *is a sole proprietorship, then the number of staff must equal zero. This may be expressed as* $(t \in \{sp\}) \Rightarrow (p = 0)$ *or equivalently* $(t \in \{other\}) \vee (p = 0)$*. For a record* $\boldsymbol{r}^0 = (sp, p^0, c^0)$*, the error localization problem takes the following form.*

$$\underset{(\boldsymbol{r},\boldsymbol{\Delta},\ell)\in D_t\times\mathbb{R}^2\times\{0,1\}^4}{\text{Minimize}} \; \Delta_t + \Delta_p + \Delta_c$$

$$\text{s.t.} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -M \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -M & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -M & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -M & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -M & 0 \end{bmatrix} \begin{bmatrix} d_{sp}(t) \\ d_{other}(t) \\ p \\ c \\ \Delta_t \\ \Delta_p \\ \Delta_c \\ \ell \end{bmatrix} \begin{matrix} \geqslant \\ \geqslant \\ \geqslant \\ \geqslant \\ = \\ > \\ = \\ \leqslant \\ \leqslant \\ \leqslant \\ \leqslant \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ p^0 \\ -p^0 \\ c^0 \\ -c^0 \end{bmatrix}$$

The first row in the restriction represents the mixed-type rule, translated as shown in Eq. (28)*. Row six connects the indicator variable* ℓ *with the numerical edit in the consequent of* $t \in \{sp\} \Rightarrow (p = 0)$*. Rows two, three and four represent the numerical edits limiting values of* p *and* c*. Row five forces* t *to have only one value and row seven connects the value of* t *with that of* Δ_t*. Finally, rows eight to eleven connect the numerical variables with the corresponding change indicators.* □

3 Numerical stability issues

An error localization problem, in its original formulation, is an optimization problem over n binary decision variables that indicate which variables in a record should be adapted. Depending on the type of rules, its reformulation as a MIP problem adds at least n variables and $2n$ restrictions. Moreover, the reformulation as a MIP problem introduces a constant M, the value of which has no mathematical significance but for which a value must be chosen in practice. Because of limitations in machine accuracy, which is typically on the order of 10^{-16}, the range of problems that can be solved is limited as well. In particular, MIP problems that involve both very large and very small numbers in the objective function and/or the restriction matrix may yield erroneous solutions or become numerically unfeasible. Indeed, the manual of `lp_solve` (Berkelaar et al., 2010) points out that `*[...] to improve stability, one must try to work with numbers that are somewhat in the same range. Ideally in the neighborhood of 1*'. The following subsections point out a number of sources of numerical instabilities and provides ways to handle them.

3.1 A short overview of MIP-solving

Consider a set of linear restrictions on numerical data of the form $\boldsymbol{Ax} \leq \boldsymbol{b}$, where we assume $\boldsymbol{b} \geq \boldsymbol{0}$ and the restrictions consist solely of inequalities ($\leq$). In practice, these restrictions will not limit the type of linear rules covered by this discussion, since it can be shown that all linear rules can be brought to this form, possibly by introducing dummy variables [see *e.g.* Schrijver (1998); Bradley et al. (1977)]. Furthermore, suppose we have a record $\boldsymbol{x}^0 \geq \boldsymbol{0}$ which doesn't obey the restrictions. The MIP formulation of the error localization problem can be written as follows.

$$\begin{aligned} &\text{Minimize } f = \boldsymbol{w}^T\boldsymbol{\Delta} \\ &\text{s.t.} \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \mathbb{1} & -\boldsymbol{M} \\ -\mathbb{1} & -\boldsymbol{M} \\ \boldsymbol{0} & \mathbb{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\Delta} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{x}^0 \\ -\boldsymbol{x}^0 \\ \boldsymbol{1} \end{bmatrix}, \end{aligned} \tag{31}$$

and $\boldsymbol{x}, \boldsymbol{\Delta} \geq \boldsymbol{0}$. Also, $\mathbb{1}$ denotes the unit matrix, $\boldsymbol{1}$ a vector with all coefficients equal to 1 and $\boldsymbol{M} = \mathbb{1}M$. The last row is added to force $\boldsymbol{\Delta} \leq \boldsymbol{1}$. This is necessary because we will initially treat the binary variables Δ_j as if they are real numbers in the range $[0, 1]$.

The `lp_solve` library uses an approach based on the revised Phase I - Phase II simplex algorithm to solve MIP problems. In this approach every inequality of Eq. (31) is transformed to an equality by adding dummy variables: each row $\boldsymbol{a}^T\boldsymbol{x} \leq b$ is replaced by $\boldsymbol{a}^T\boldsymbol{x} + s = b$, with $s \geq 0$. Depending on the sign of the inequality, the extra variable s is called a *slack* or *surplus* variable. In Eq. (31) there are four sets of restrictions (rows). We therefore need to add four sets of surplus and slack variables (columns) in order to rewrite the whole system in terms of equalities.

Note that after this transformation, the whole problem including the cost function is written in terms of equalities. It is customary to organize this set of equality objective function in a single *tableau* notation as follows.

$$\left[\begin{array}{c|cccccc|c} 1 & \boldsymbol{0} & -\boldsymbol{w}^T & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & 0 \\ \hline 0 & \boldsymbol{A} & \boldsymbol{0} & \mathbb{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{b} \\ 0 & \mathbb{1} & -\boldsymbol{M} & \boldsymbol{0} & \mathbb{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{x}^0 \\ 0 & \mathbb{1} & \boldsymbol{M} & \boldsymbol{0} & \boldsymbol{0} & -\mathbb{1} & \boldsymbol{0} & \boldsymbol{x}^0 \\ 0 & \boldsymbol{0} & \mathbb{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \mathbb{1} & \boldsymbol{1} \end{array}\right]. \tag{32}$$

Here, the first row and column represents the cost function. Columns two and three correspond to the original set of variables in Eq. (31) while columns four to seven correspond to sets of slack and surplus variables. The final column contains the constant vector.

A tableau representation shows all the numbers that are relevant in an LP-problem at a glance. By examining how LP-solvers typically manipulate these numbers we gain some insight into how and where numerical stability issues may arise.

Since the tableau represents a set of linear equalities, it may be manipulated as such. In fact, the simplex method is based on performing a number of cleverly chosen Gauss-Jordan elimination steps on the tableau. For a complete discussion the reader is referred to one of the many textbooks discussing it (e.g. Bradley et al. (1977)), but in short the Phase I - Phase II simplex algorithm consists of the following steps.

Phase I: Repeatedly apply Gauss-Jordan elimination steps (called *pivots*) to derive a decision vector that obeys all restrictions. A vector obeying all restrictions is called a *basic solution*.

Phase II: Repeatedly apply pivots to move from the initial non-optimal solution to the solution that minimizes the objective function f.

In Phase I, a decision vector $(\boldsymbol{x}, \boldsymbol{\Delta}, \boldsymbol{s})$ (with $\boldsymbol{s}$ the vector of slack and surplus variables) is derived that obeys all restrictions. The precise algorithm need not be described here. It involves adding again extra variables where necessary and then manipulating the system of equalities represented by the tableau so that those extra variables are driven to zero. The binary variables $\boldsymbol{\Delta}$ are first treated as if they are real variables. In the Appendix it is shown in detail how an initial solution for Eq. (32) can be found, here we just state the result of a Phase-I operation:

$$\left[\begin{array}{c|cccccc|c} 1 & \boldsymbol{w}^T\boldsymbol{M}^{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\boldsymbol{w}^T\boldsymbol{M}^{-1} & \mathbf{0} & \boldsymbol{w}^T\boldsymbol{M}^{-1}\boldsymbol{x}^0 \\ \hline 0 & \boldsymbol{A} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{b} \\ 0 & 2\mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbb{1} & -\mathbb{1} & \mathbf{0} & 2\boldsymbol{x}^0 \\ 0 & \boldsymbol{M}^{-1} & \mathbb{1} & \mathbf{0} & \mathbf{0} & -\boldsymbol{M}^{-1} & \mathbf{0} & \boldsymbol{M}^{-1}\boldsymbol{x}^0 \\ 0 & -\boldsymbol{M}^{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{M}^{-1} & \mathbb{1} & \mathbf{1} - \boldsymbol{M}^{-1}\boldsymbol{x}^0 \end{array}\right]. \tag{33}$$

This tableau immediately suggests a valid solution: it is easily confirmed by matrix multiplication that the vector $(\boldsymbol{x}, \boldsymbol{\Delta}, \boldsymbol{s}) = (\mathbf{0}, \boldsymbol{x}^0\boldsymbol{M}^{-1}, [\boldsymbol{b}, 2\boldsymbol{x}^0, \mathbf{0}, \mathbf{1} - \boldsymbol{M}^{-1}\boldsymbol{x}^0])$ obeys all restrictions. The above form of a tableau, where the restriction matrix contains a (column permutation of) the unit matrix, the right-hand-side has only non-negative coefficients, and the cost vector equals zero for the columns above the unit matrix is called the *canonical form*.

Now, a *pivot* operation consists of the following steps:

1. Select a positive element R_{ij} from the restriction matrix. This is called the *pivot element*.
2. Multiply the ith row by R_{ij}^{-1}.
3. Subtract the ith row, possibly after rescaling, from all other rows of the tableau such that their jth column equals zero.

The result of a pivot operation is again a tableau in canonical form but with possibly a different value for the cost function. The simplex algorithm proceeds by selecting pivots that decrease the cost function until the minimum is reached or the problem is shown to be unfeasible.

Up until this point, we have treated the binary variables $\boldsymbol{\Delta}$ as if they were real variables, so the tableaux discussed above do not represent solutions to our original problem which demands that all Δ_j are either 0 or 1. In the `lp_solve` library this is solved as follows.

1. For each optimized value Δ_j^* test whether it is 0 or 1. If all Δ_j^* are integer, we have a valid solution of objective value $\boldsymbol{w}^T\boldsymbol{\Delta}$ and we are done.
2. For the first variable Δ_j^* that is not integer, create two sub-models: one where the minimum value of Δ_j equals 1 and one where the maximum value of Δ_j equals 0.
3. Optimize the two sub-models. If solutions exist, the result will contain an integer Δ_j.
4. For the sub-models that have a solution and whose current objective value does not exceed that of an earlier found solution, return to step 1.

The above *branch-and-bound* approach completes this overview. The discussion of pivot and branch-and-bound operations has so far been purely mathematical: no choices have been made regarding issues such as how to decide when the floating-point representation of a value is regarded zero or how to handle badly scaled problems. Do note however, that in the course of going from Phase-I to Phase-II, the LP-solver is handling numbers that may range from M^{-1} to M which typically differ many orders of magnitude

3.2 Scaling numerical records

In the MIP formulation of error localization over numerical records under linear restrictions, Eq. (7) restricts the search space around the original value x^0 to $|x - x^0| \leq M$. This restriction may prohibit a MIP solver from finding the actual minimal set of values to adapt or even render the MIP-problem unsolvable. As an example, consider the following error localization problem on a two-variable record.

$$\begin{cases} x_1 \geq x_2 \\ \boldsymbol{x}^0 = (10^6, 10^9). \end{cases}$$

Obviously, the record can be made to obey the restriction by multiplying x_1^0 by 10^3 or by dividing x_2^0 by the same amount. However, in `editrules` the default value for $M = 10^7 < 10^9 - 10^6$ which renders the corresponding MIP problem unsolvable. Practical examples where such errors occur is when a value is recorded in the wrong unit of measure (*e.g.* in € instead of k€.).

It is therefore advisable to remove such unit-of-measure errors prior to error localization[‡)] and to express numerical records on a scale such that all $|x^0| \ll M$. Note that under linear restrictions (Eq. 5) one may always apply a scaling factor $k > 0$ to a numerical record $\boldsymbol{x}$ by replacing $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ with $\boldsymbol{A}(k\boldsymbol{x}) \leq k\boldsymbol{b}$. In the above example, one may replace $\boldsymbol{x}^0$ by $10^{-6}\boldsymbol{x}^0$ for the purpose of error localization. If $\boldsymbol{b} = \boldsymbol{0}$ and the coefficients of $\boldsymbol{x}$ do not vary over many orders of magnitude, such a scaling will suffice to numerically stabilize the MIP problem.

3.3 Setting numerical threshold values

On most modern computer systems real numbers are represented in IEEE (2008) double precision format. In essence, real numbers are represented as rounded-off fractions so arithmetic operations on such numbers always result in loss of precision and round-off errors. For example, even though mathematically we have $0.7 - 0.5 = 0.2$, in the floating point representation (denoted $\text{fl}(\cdot)$) we have $\text{fl}(0.7) - \text{fl}(0.5) \neq \text{fl}(0.2)$. In fact, the difference is about $0.56 \cdot 10^{-16}$ in this case.

This means that in practice one cannot rely on equality tests to determine whether two floating point numbers are equal. Rather, one considers two numbers v and w equal when $|\,\text{fl}(v) - \text{fl}(w)|$ is smaller than a predefined tolerance. For this reason `lp_solve` comes with a number of predefined tolerances. These tolerances have default values but these may be altered by the user.

The tolerances implemented by `lp_solve` are summarized in Table 3.1. The value of `epspivot` is used to determine whether an element of the restriction matrix is positive so it may be used as a pivoting element. Its default value is $2 \cdot 10^{-7}$, but note that after Phase I, our restriction matrix contains elements on the order of $M^{-1} = 10^{-7}$. For this reason, the value of `epspivot` is lowered in `editrules` by default, but users may override these settings. For the same reason, the value of `epsint`, which determines when a value for one of the $\boldsymbol{\Delta}_j$ can be considered integer is lowered in `editrules` as well. The other tolerance settings of `lp_solve`: `epsb` (to test if the right-hand-side of the restrictions differ from 0), `epsd` (to test if two values of the objective

[‡)] Methods for detecting such errors exist, see for example De Waal et al. (2011), Chapter 2. In fact, the principle of minimal change is not applicable here since a better value can be deduced from the cause of the error.

Table 3.1 Numerical parameters for MIP based error localization.

	Default value		
Parameter	`lp_solve`	`editrules`	meaning
`M`	–	10^{7}	set bounds so $\boldsymbol{x} \in \boldsymbol{x}^{0} \pm M$
`eps`	–	10^{-3}	translate $x < 0$ to $x \leq \varepsilon$
`epspivot`	$2 \cdot 10^{-7}$	10^{-15}	test if pivot element $R_{ij} > 0$
`epsint`	10^{-7}	10^{-15}	test if $\Delta_j \in \mathbb{N}$
`epsb`	10^{-10}	10^{-10}	test if $b_i > 0$
`epsd`	10^{-9}	10^{-9}	test if obj. values $\lvert f - f' \rvert > 0$ during simplex
`epsel`	10^{-12}	10^{-12}	test if other numbers $\neq 0$
`mip_gap`	10^{-11}	10^{-11}	test if obj. values $\lvert f - f' \rvert > 0$ during B&B

function differ), `epsel` (all other values) and `mip_gap` (to test whether a bound condition has been hit in the branch-and-bound algorithm) have not been altered.

The limited precision inherent to floating point calculations imply that computations get more inaccurate as the operands differ more in magnitude. For example, on any system that uses double precision arithmetic the difference $\mathrm{fl}(1) - \mathrm{fl}(10^{-17})$ is indistinguishable from $\mathrm{fl}(1)$. This then, leads to two contradictory demands on our translation of an error localization problem to a MIP problem. On one hand, one would like to set M as large as possible so the ranges $x_j^0 \pm M$ contain a valid value of x_j. On the other hand, large values for M imply that MIP problems such as Eq. (31) may become numerically unstable.

In practice, the tableau used by `lp_solve` will not be exactly the same as represented in Eq. (33). Over the years, many optimizations and heuristics have been developed to make solving linear programming problems fast and reliable, and several of those optimizations have been implemented in `lp_solve`. However, the tableau of Eq. (33) does fundamentally show how numerical instabilities may occur: the tableau simultaneously contains numbers on the order of M^{-1} and on the order of $\boldsymbol{x}^0$. It is not at all unlikely that the two differ in many orders of magnitude.

The above discussion suggests the following rules of thumb to avoid numerical instabilities in error localization problems.

1. Make sure that elements of $\boldsymbol{x}^0$ are expressed in units such that $\boldsymbol{A}$, $\boldsymbol{b}$ and $\boldsymbol{x}$ are on the order of 1 wherever possible.
2. Choose a value of M appropriate for $\boldsymbol{x}^0$.
3. If the above does not help in stabilizing he problem, try lowering the numerical constants of Table 3.1.

In our experience, the settings denoted in Table 3.1 have performed well in a range of problems where elements of $\boldsymbol{A}$ and $\boldsymbol{b}$ are on the order of 1 and values of $\boldsymbol{x}^0$ are in the range $[1, 10^8]$. However, these settings have been made configurable so users may choose their own settings as needed.

4 Usage

In the `editrules` package, edits can be defined with the `editset` function. For example, the command

```
> E <- editset(expression(
+    x + y == z
+   , if ( x > y ) y > 0
+ ))
```

defines two edits on the variables x, y and z and stores them in an object called `E`. Here, `E` is an object of type `editset` and it can be used to store and manipulate linear (in)equality edits, edits on categorical data as well as edits on mixed-type data. Besides `editset` there are specialized functions called `editmatrix` and `editarray` which can be used to define rules on purely numerical or purely categorical data respectively. Edits are defined in basic `R` syntax; one may use multiplication, addition, `if-else` statements, logical operators and the `%in%` operator for set inclusion on categorical variables.

Besides defining rules on the command line, as in the example above, one may store the rules in a text file and read the rules into `R` using the `editfile` function.

```
> E <- editfile("myedits.txt")
```

Here, `myedits.txt` is the name of a textfile containing the edits. The resulting object is by default of class `editset`. If the extra argument `type="num"` or `type="cat"` is passed, only numerical or categorical edits are read from the file. For a further discussion of the functions mentioned here we refer the reader to the technical manual that is included with the software. Also see the papers of Van der Loo and De Jonge (2011, 2012) for a precise description of the edit-definition syntax.

4.1 Error localization

The main interface to error localization functionality is the `localizeErrors` function. The function accepts editrules in the form of an `editset`, `editmatrix` or `editarray` object, and a data set in the form of a `data.frame`. By default the function localizes errors using the branch-and-bound algorithm. One may switch to the MIP-based approach by setting the parameter `method="mip"` as in the following example.

```
> E <- editmatrix("x <= y")
> dat <- data.frame(x = c(10, 3), y = c(1, 5))
> el <- localizeErrors(E, dat, method = "mip")
```

The object returned by `localizeErrors` contains the error locations as well as some details on how the algorithm ran. The error locations are stored in a boolean array called `adapt` which can be accessed as follows.

```
> el$adapt
      x     y
1  TRUE FALSE
2 FALSE FALSE
```

Here, the array has dimension 2×2 since the input data set consisted of two records with two variables. Here, the result indicates for the first record that by altering the value for x, the record can be corrected to obey the edit rule stored in `E`. For the second record, no alterations are necessary.

Details on the error localization procedure are stored in a `data.frame` called `status`, which can be accessed as follows.

```
> el$status
  weight degeneracy user system elapsed maxDurationExceeded memfail
1      1         NA 0.04      0    0.05               FALSE   FALSE
2      0         NA 0.02      0    0.01               FALSE   FALSE
```

The status `data.frame` contains one record of information for each record in the input data. The column `weight` contains the value of the objective function as defined by Eq. (2). The time it took to perform the calculation is subdivided into user, system and elapsed time, where the latter corresponds to the actual time that has passed on the clock. The boolean variable (`maxDurationExceeded`) indicates whether the time limit for finding a solution was exceeded. There are two status columns that have no relevance when the error localization method is MIP. First, the indicator `memfail` can only be set to `TRUE` when the branch-and-bound algorithm is used. It indicates that perhaps the optimal solution could not be found because of memory limitations. Second, and more importantly, the `degeneracy` parameter is not set when `method="mip"`. This parameter indicates how many equivalent solutions there are to each error localization problem. Contrary to the branch-and-bound method for error localization, the MIP-based approach does not return this information.

The output of `localizeErrors` can be controlled with two parameters. Most importantly, positive weights for each variable and optionally for each record can be set; variables with lower weights attached to them are more likely to be part of a chosen solution which is otherwise degenerate. Furthermore, a maximum search time per record can be specified, the default setting being 10 minutes. Finally, the optional parameter `lpcontrol` may contain a list of parameters to be passed to `lpSolveAPI`. The default settings can be listed as follows

```
> options("er.lpcontrol")
$er.lpcontrol
NULL
```

These options are precisely the values that differ from `lpSolve`'s default settings listed in Table 3.1. Changing or adding options can be done either by passing the `lpcontrol` parameter to `localizeErrors`, in which case it is only used for the current error localization problem. To adapt a parameter for the remainder of the running R-session or until the option is reset one can use `R`'s `options` function. For example, to alter the `epsb` parameter through `localizeErrors`, one may use either of the two calls below.

```
> localizeErrors( E, dat, lpcontrol = c(epsb=1e-12, options("er.lpcontrol")) )
> options( er.lpcontrol = c(options("er.lpcontrol"), epsb=1e=12) )
```

The important thing to note is that it is up to the user to merge new options with existing ones since `lpcontrol` completely overwrites the default settings. A precise description of possible options is also given in the reference manual of `lpSolveApi`.

4.2 Fine-grained control options

For users who wish to exert more fine-grained control on the MIP-solver or who wish to interface editrules with another MIP-solving engine, two lower-level functionalities have been exposed to the user.

The first is errorLocalizer_mip. This function takes a set of edit rules in the form of an editset, editmatrix or editarray and a single data record in the form of a named list.

```
> L <- errorLocalizer_mip(E, list(x = 10, y = 1))
```

Here, errorLocalizer_mip translates the error localization problem for a single record to a MIP problem, feeds it to lpSolveAPI and returns all the results in a list. The list contains two extra pieces of information not available in the output of localizeErrors. The first is a parameter called x_feasible, containing a record that actually obeys all the edit rules.

```
> L$x_feasible
$x
[1] 1

$y
[1] 1
```

The second parameter is called lp. This is an object of class lpExtPtr which points to an object of lpSolveApi, stored outside of R's memory. It contains precise information on the definition of the MIP problem as interpreted by lpSolveAPI. It can be manipulated or exported to a text file using write.lp of the lpSolveAPI package.

The second functionality entails the functions as.mip and as.lp.mip. The function as.mip allows users to translate the combination of a set of editrules and a data record to a MIP problem.

```
> mip <- as.mip(E, list(x = 10, y = 1))
> print(mip)
num1 : x <= y
x0   : x <= 1e+07*delta.x + 10
y0   : y <= 1e+07*delta.y + 1
x0_1 : 10 <= 1e+07*delta.x + x
y0_1 : 1 <= 1e+07*delta.y + y
objective function = min:  1*delta.x + 1*delta.y
```

The object returned by as.mip can be used to inspect how editrules translates an error localization problem to a MIP problem. The interested reader may want to compare the above representation with Eqs. (2), (6) and (7).

This representation of the MIP problem can be translated to a form that is suited for solving with lpSolveAPI.

```
> lp <- as.lp.mip(mip)
```

The lp object can directly be used as input for lpSolveAPI or written to disk with write.lp as follows.

```
> write.lp(lp, file = "myLPfile.lp")
```

This command produces a text file that is written in a syntax understood by the lp_solve commandline program. lp_solve also has facilities to translate this syntax to other formats allowing export to other LP-solvers, including some commercial ones.

5 Benchmarks

The *branch and bound* and *MIP*-based algorithms for error localization differ in consumption of both memory and computational time. The performance of error localization methods depends on the number of variables, the number of erroneous fields, the number of violated restrictions, and the total number of restrictions.

Below we describe benchmarks based edit sets which can be systematically extended to encompass more variables. The edit sets were designed so that synthetic records can be created where the size of the solution to the error localization problem (the minimal number of fields to alter so the record can be made consistent) can be easily controlled as well. This then gives an impression of computational time consumed by the MIP and branch-and-bound approaches as a function of the number of variables/edits and the number of erroneous fields.

Benchmarking the two approaches is further complicated by the fact that performance of especially the branch-and-bound algorithm is strongly affected by the order in which variables are treated by the algorithm. Indeed, preliminary tests showed that if the erroneous variables are treated first (which may be achieved by setting appropriate reliability weights), performance of the branch-and-bound method is largely on par with that of the MIP approach. If variables are ordered such that erroneous variables are treated halfway, or at the end of the variable set, the branch-and-bound method performs much slower than the MIP method. Below, we report on benchmarks where errors were injected into variables which were positioned around the center of the record. This mimics the case when there is no knowledge on the reliability of the variables.

5.1 Linear restrictions

The edits used in this benchmark form a balance system. Balance systems occur for example in energy or business statistics where main variables (total energy consumption, total turnover) are the sum of several other variables. Moreover, these main variables are also connected by linear restrictions (total energy production equals total energy consumption). In balance systems variables are therefore connected through a tree-like structure where the value corresponding to a node equals the sum of its child node values.

In our benchmark we generate a balance system on $2n + 1$ variables. Here, the restrictions connect the variables through a binary tree where the value of a node is the sum of its two child node values. The value of the top node is restricted to be non-negative and always at least as large as any other value.

$$\begin{aligned} x_1 &= x_2 + x_3 \\ &\vdots \\ x_n &= x_{2n} + x_{2n+1} \\ x_1 &\geq 0 \\ x_1 &\geq x_i, \quad i = 2, 3, \ldots 2n+1 \end{aligned}$$

This edit set is fully connected and completely fixed by choosing an $n > 0$. The $x_1 \geq 0$ and $x_1 \geq x_i$ for all $2 \leq i \leq 2n + 1$ implies that all $x_i \geq 0$. To see this, observe that since $x_1 \geq 0$ and

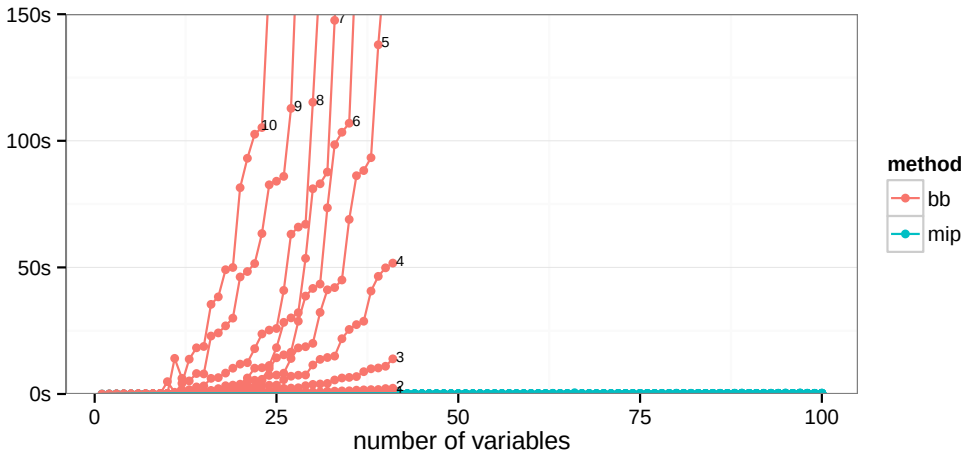

Figure 5.1 Linear edits, each corresponds to a different number of errors

and $x_1 \geq x_3$, we cannot have $x_2 < 0$ without having $x_3 > x_1$, which violates the restrictions saying that x_1 must be larger than or equal to x_i, $i > 1$. This reasoning applies recursively to all x_i because of the chained sum-rules.

One solution to this system is the zero vector $(x_1, \ldots, x_{2n+1}) = (0, \ldots, 0)$. An error can be introduced in the data by setting some x_i to -1. Because of the implied nonnegativity contstraint, the number of variables in the optimal error localization solution is exactly equal to the number of x_i set to -1. The benchmark was performed for balance systems with 1 to 101 variables ($n = 0$ to 50). For each system records with 1 to 10 errors were generated.

Figure 5.1 shows the time of error localization for increasing number of variables and increasing number of errors, both for the branch-and-bound method and the MIP method. The branch-and-bound method hits the `exponential wall' around 30 variables and 5 errors or 20 variables with 10 errors, showing reasonable performance only when the number of errors is 2 or less. The branch-and-bound algorithm was broken off when no solution was found in less than 10 minutes which occurred at problems with more than 50 variables and 4 errors. In contrast, the MIP-based approach performs well (under a few seconds) for all problems tested here.

We caution the reader to conclude that the MIP approach is better performing in all circumstances: when a good set of reliability weights can be determined, variables that are most likely to be erroneous can be treated early on by the algorithm, yielding a considerable performance boost. In such a case, the optimal solution is found first by the algorithm and branches leading to suboptimal solutions can be quickly rejected. However, in the generic case where no reliability weights can be derived with great confidence, the MIP-approach is obviously better performing.

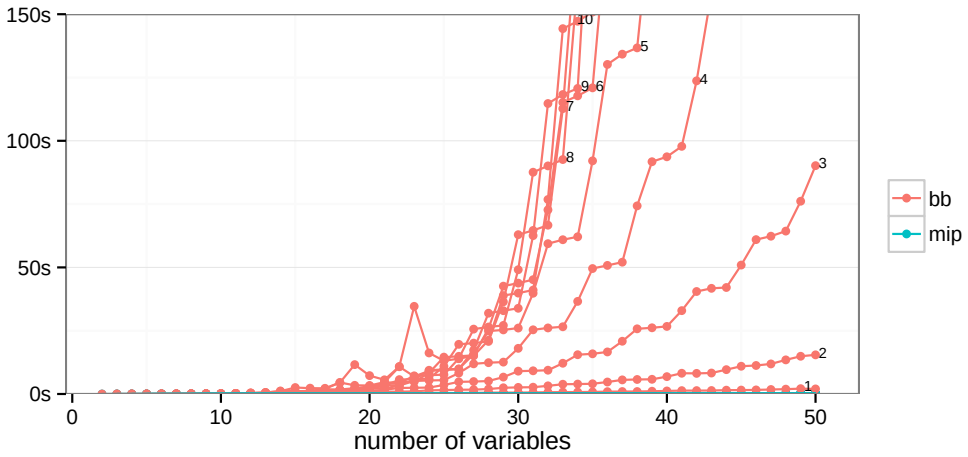

Figure 5.2 Categorical edits: each line corresponds to a different number of errors.

5.2 Categorical restrictions

For this benchmark we use a chain of interconnected edits:

$$\begin{aligned} & v_1 == \texttt{TRUE} \\ \texttt{if } (v_1 == \texttt{TRUE}) \; & v_2 == \texttt{TRUE} \\ & \vdots \\ \texttt{if } (v_{n-1} == \texttt{TRUE}) \; & v_n == \texttt{TRUE}. \end{aligned}$$

Examples of such edit chains occur in practice, for example: `if` *married*==`TRUE then` *adult*==`TRUE` and `if` *adult*==`TRUE then` *allowed to drive a car*==`TRUE`. Here, we demand that $v_1 ==$ `TRUE` which forces the only solution to this editset to be $(v_1, \ldots, v_n) =$ (`TRUE`, … , `TRUE`). An error in the data can therefore be introduced by setting on of the v_i's to `FALSE`.

Benchmarks have been performed for systems with $n = 1$ to $n = 50$ variables. For each system records with 1 to 10 errors were created. Again, the errors were introduced in variables with intermediate positions in the record.

Figure 5.2 shows time spent on error localization for the branch-and-bound and MIP-approaches as a function of number of variables and errors. Again, the MIP-based algorithm outperforms the branch-and-bound approach in nearly every case. Problems with more than 30 variables and 4 errors yield no solution with the branch-and-bound approach within 10 minutes causing calculations to be terminated. The same caution holds here as for the benchmarks on linear edits described above: carefully chosen reliability weights can improve performance of the branch-and-bound method. Without such knowledge however, MIP is the better generic choice when performance is important.

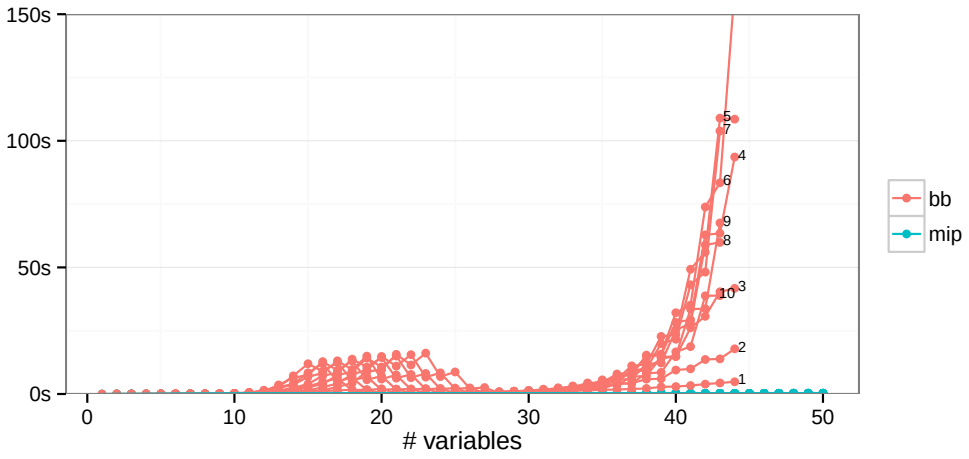

Figure 5.3 Mixed type edits: each line represents a different number of errors.

5.3 Mixed-type restrictions

We use a chain of interconnected restrictions defined as follows:

$$\begin{aligned} & & x_1 \geq 0 \\ & \texttt{if } (x_1 \geq 0) & x_2 \geq 0 \\ & & \vdots \\ & \texttt{if } (x_{N-1} \geq 0) & x_N \geq 0. \end{aligned}$$

Here, all x_i are numeric but since the conditional restrictions are internally modeled using dummy boolean variables, it serves as a model for mixed-type variable restrictions. Examples of such chains do occur in practice, for example the following restriction is often found in the context of business statistics: `if` *number of employees* > 0 `then` *amount of salary payed* > 0.

One viable solution for this system of edits is the zero vector $(x_i, \ldots, x_n) = (0, \ldots, 0)$. Note that the restriction set implies that $x_i \geq 0$ for $i = 1 \ldots N$. An error can be injected by setting one or more $x_i = -1$. The set of edits is engineered such that if k variables are set to -1, then k variables must be adapted to re-establish a viable solution. However, setting k variables to -1 does not mean that k explicitly defined edits are violated. For example, it is easily confirmed that for $n = 4$ and $\boldsymbol{x} = (0, 0, -1, -1)$ only the third (out of five) edit is violated. However both x_3 and x_4 need to be adapted in order to repair the record.

The benchmark was performed with 1 to 50 variables and 1 to 10 errors introduced. Again, errors were injected at variables with intermediate positions in the records.

The results of the benchmarks are shown in Figure 5.3. Results are comparable with the benchmarks for linear and categorical data type edits, except for the `bump' computational time for the branch-and-bound method around 10-25 variables. Interpretation is difficult without precisely following the state of variables during the run of the algorithm, but a plausible explanation is fundamental difference between the branch-and-bound approaches for

single-type and mixed-type edits. For linear and categorical edits the branch-and-bound algorithm traverses a tree that branches over the variables whereas for mixed-type edits there is also a bifurcation over each condition in the mixed-type edits. This bifurcation generates a lot of extra edits compared to the situation with single-type edits. On the other hand, branching over variables includes simplifying steps that reduce the number of edits. Tests have shown that interaction between these two effects strongly depends on the order of the variables and which variables contain the actual error. The observed `bump' should therefore be regarded an artefact of this specific benchmark. However, this does not alter the conclusion that the MIP approach performs consistently better.

6 Conclusion

We described a formulation of the error localization problem for linear, categorical and mixed-type restriction in terms of mixed-integer programming problems. It was shown that in this formulation the mixed-type restrictions can be understood as a generalisation of both linear and categorical restrictions.

Although mixed integer programming problems can be solved by readily available software packages, there may be a trade-off in numerical stability with respect to a branch-and-bound approach. This holds especially when typical default settings of such software is left unchanged and data records and/or linear coefficients of the restriction sets cover several orders of magnitude. In this paper we locate the origin of these instabilities and provide some pointers to avoiding such problems.

The `lp_solve` package has now been introduced as a MIP-solver backend to our `editrules` R package for error localization and rule management. Our benchmarks indicate that in generic cases where no prior knowledge is available about which values in a record may be erroneous (as may be expressed by lower reliability weights), the MIP method is much faster than the previously implemented branch-and-bound based algorithms. On the other hand, the branch-and-bound based approach returns extra information, most notably the number of equivalent solutions. The latter can be used as a indicator for quality of automatic data editing.

References

Berkelaar, M., K. Eikland, and P. Notebaert (2010). *lpsolve: Open source (Mixed-Integer) Linear Programming system*. Version 5.5.2.0 released 8 december, 2010.

Bradley, S., A. Hax, and T. Magnanti (1977). *Applied mathematical programming*. Addison-Wesley.

De Jonge, E. and M. Van der Loo (2011). Manipulation of linear edits and error localization with the editrules package. Technical Report 201120, Statistics Netherlands, The Hague.

De Jonge, E. and M. van der Loo (2011-2013). *editrules: An R package for parsing and manipulating edit rules and error localization.*

De Waal, T. (2003). *Processing of erroneous and unsafe data*. Ph. D. thesis, Erasmus University Rotterdam.

De Waal, T., J. Pannekoek, and S. Scholtus (2011). *Handbook of statistical data editing and imputation*. Wiley handbooks in survey methodology. John Wiley & Sons.

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.

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

IEEE (2008). IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, 1--58.

Konis, K. (2011). *lpSolveAPI: R Interface for lpsolve version 5.5.2.0*. R package version 5.5.2.0-5.

Schrijver, A. (1998). *Theory of linear and integer programming*. Wiley-Interscience series in descrete mathematics and optimization. New York: John Wiley and Sons.

Van der Loo, M. and E. De Jonge (2011). Manipulation of categorical data edits and error localization with the editrules package. Technical Report 201129, Statistics Netherlands.

Van der Loo, M. and E. De Jonge (2012). Manipulation of conditional restrictions and error localization with the editrules package. Technical Report 2012XX, Statistics Netherlands. In press.

Van der Loo, M., E. De Jonge, and S. Scholtus (2011). Correction of rounding, typing and sign errors with the deducorrect package. Technical Report 201119, Statistics Netherlands. R package version 1.0-0.

Winkler, W. E. (1999). State of statistical data editing and current research problems. In *Working paper no. 29. UN/ECE Work Session on Statistical Data editing*, Rome.

Appendix

I Derivation of Equation 33

In the Phase I Phase II simplex method, phase I is aimed to derive a valid solution which is then iteratively updated to an optimal solution in Phase II. Here, we derive a Phase I solution, specific for error localization problems.

Recall the tableau of Eq. (32); for clarity, the top row indicates to what variables the columns of the tableau pertain.

$$\left[\begin{array}{c|cccccc|c} f & \boldsymbol{x}^T & \boldsymbol{\Delta} & \boldsymbol{s}_x & \boldsymbol{s}_+ & \boldsymbol{s}_- & \boldsymbol{s}_\Delta & \\ 1 & \mathbf{0} & -\boldsymbol{w}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 \\ \hline 0 & \boldsymbol{A} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{b} \\ 0 & \mathbb{1} & -\boldsymbol{M} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \boldsymbol{x}^0 \\ 0 & \mathbb{1} & \boldsymbol{M} & \mathbf{0} & \mathbf{0} & -\mathbb{1} & \mathbf{0} & \boldsymbol{x}^0 \\ 0 & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbb{1} & \mathbf{1} \end{array}\right].$$

Here, the $\boldsymbol{s}_i$ are slack or surplus variables, aimed to write the original inequality restrictions as equalities. The $\boldsymbol{s}_x$ are used to rewrite restrictions on observed variables, the $\boldsymbol{s}_\pm$ to write the upper and lower limits on $\boldsymbol{x}$ as equalities and $\boldsymbol{s}_\Delta$ to write the upper limits on $\boldsymbol{\Delta}$ as equalities.

Observe that the above tableau *almost* suggests a trivial solution. If we choose $\boldsymbol{s}_x = \boldsymbol{b}$, $\boldsymbol{s}_\pm = \pm\boldsymbol{x}^0$ and $\boldsymbol{s}_\Delta = \mathbf{1}$, we may set $(\boldsymbol{x}, \boldsymbol{\Delta}) = \mathbf{0}$. However, recall that we demand all variables to be non-negative so $\boldsymbol{s}_-$ may not be equal to $-\boldsymbol{x}^0$. To resolve this problem, we introduce a set of *artificial* variables $\boldsymbol{a}_-$ and extend the restrictions involving $\boldsymbol{s}_-$ as follows:

$$\boldsymbol{x} + M\mathbf{1} - \boldsymbol{s}_- + \boldsymbol{a}_- = \boldsymbol{x}_0.$$

This yields the following tableau.

$$\left[\begin{array}{c|cccccccc|c} f & \boldsymbol{x}^T & \boldsymbol{\Delta} & \boldsymbol{s}_x & \boldsymbol{s}_+ & \boldsymbol{s}_- & \boldsymbol{s}_\Delta & \boldsymbol{a}_- & \\ 1 & \mathbf{0} & -\boldsymbol{w}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 \\ \hline 0 & \boldsymbol{A} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{b} \\ 0 & \mathbb{1} & -\boldsymbol{M} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{x}^0 \\ 0 & \mathbb{1} & \boldsymbol{M} & \mathbf{0} & \mathbf{0} & -\mathbb{1} & \mathbf{0} & \mathbb{1} & \boldsymbol{x}^0 \\ 0 & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{1} \end{array}\right].$$

The essential point to note is that the tableau now contains a unit matrix in columns 4, 5, 7 and 8, so choosing $\boldsymbol{s}_x = \boldsymbol{b}$, $\boldsymbol{s}_+ = \boldsymbol{x}^0$, $\boldsymbol{s}_\Delta = \mathbf{1}$ and $\boldsymbol{a}_- = \boldsymbol{x}^0$, and $(\boldsymbol{x}, \boldsymbol{\Delta}, \boldsymbol{s}_-) = \mathbf{0}$ is a solution obeying all restrictions. The artificial variables have no relation to the original problem, so we want them to be zero in the final solution. Since the tableau represents a set of linear equalities, we are allowed to multiply rows with a constant and add and subtract rows. If this is done in such a way that we are again left with a unit matrix in part of the columns, a new valid solution is generated. Here, we add the fourth row to row three and subtract $\boldsymbol{M}^{-1}$ times the fourth row from the last row. Row four is multiplied with $\boldsymbol{M}^{-1}$. This gives

$$\left[\begin{array}{c|cccccccc|c} f & \boldsymbol{x}^T & \boldsymbol{\Delta} & \boldsymbol{s}_x & \boldsymbol{s}_+ & \boldsymbol{s}_- & \boldsymbol{s}_\Delta & \boldsymbol{a}_- & \\ 1 & \mathbf{0} & -\boldsymbol{w}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 \\ \hline 0 & \boldsymbol{A} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{b} \\ 0 & 2\mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbb{1} & \mathbf{0} & \mathbf{0} & \mathbb{1} & 2\boldsymbol{x}^0 \\ 0 & \boldsymbol{M}^{-1} & \mathbb{1} & \mathbf{0} & \mathbf{0} & -\boldsymbol{M}^{-1} & \mathbf{0} & \boldsymbol{M}^{-1} & \boldsymbol{M}^{-1}\boldsymbol{x}^0 \\ 0 & -\boldsymbol{M}^{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{M}^{-1} & \mathbb{1} & -\boldsymbol{M}^{-1} & \mathbf{1} - \boldsymbol{M}^{-1}\boldsymbol{x}^0 \end{array}\right].$$

This almost gives a new valid solution, except that the first row, representing the objective function has not vanished at the third column. However, we may re-express the objective function in terms of the other variables. Namely using row four, we have

$$\boldsymbol{w}^T\boldsymbol{\Delta} = \boldsymbol{w}^T\boldsymbol{M}^{-1}\boldsymbol{x}^0 + \boldsymbol{w}^T\boldsymbol{M}^{-1}(\boldsymbol{s}_- - \boldsymbol{x} - \boldsymbol{a}_-).$$

Substituting this equation in the top row of the tableau shows that the solution $\boldsymbol{\Delta} = \boldsymbol{M}^{-1}\boldsymbol{x}^0$, $\boldsymbol{s}_x = \boldsymbol{b}$, $\boldsymbol{s}_+ = 2\boldsymbol{x}^0$ and $\boldsymbol{s}_\Delta = \mathbf{1} - \boldsymbol{M}^{-1}\boldsymbol{x}^0$, and $(\boldsymbol{x}, \boldsymbol{s}_-, \boldsymbol{a}_-) = \mathbf{0}$ represents a valid solution. Since the artificial variables have vanishing values, we may now delete the corresponding column, and arrive at the tableau of Eq. (33).

Publisher
Statistics Netherlands
Henri Faasdreef 312, 2492 JP The Hague
www.cbs.nl

Prepress: Statistics Netherlands, Grafimedia
Design: Edenspiekermann

Information
Telephone +31 88 570 70 70, fax +31 70 337 59 94
Via contact form: www.cbs.nl/information

Where to order
verkoop@cbs.nl
Fax +31 45 570 62 68
ISSN 1572-0314

60083201407 X-10