Discussion Paper

# Simplifying constraints in data editing

2015 | 18

Jacco Daalmans

# Content

1. **Introduction** 4

2. **Simplification features** 6
   2.1 Edits 6
   2.2 Procedure 8

3. **Algorithms** 13
   3.1 Mixed Integer Programming 13
   3.2 Determining feasibility 13
   3.3 Identifying fixed values 13
   3.4 Conditional edit simplification 14
   3.5 Redundant constraint removal 17
   3.6 Partially inconsistency localisation 18
   3.7 Minimum subset of edits to be removed 19
   3.8 Identifying a small IIS 20

4. **Applications** 22

5. **Discussion** 24

6. **References** 26

**Appendix A. Edit rules as MIP-constraints** 28

**Appendix B. A specific redundant edit** 29

**Appendix C. Bruni and Bianchi edit set** 30

## Abstract

Data editing is the process of checking and correcting data. In practise, these processes are often automated. A large number of constraints needs to be handled in many applications. This paper shows that data editing can benefit from constraint simplification techniques that are often used in Operations Research and Artificial Intelligence. Performance can be improved and a better quality of automatically corrected data can be obtained. First, a new procedure for constraint simplification will be proposed that is especially developed for data editing; a procedure that combines several known algorithms from Operations Research and Artificial Intelligence. Thereafter, it will be demonstrated that real-life edit sets can actually be simplified.

# 1. Introduction

Statistical offices have the task of publishing indisputable information about a diversity of subjects. Unfortunately, collected micro data usually contain errors, e.g. pregnant men, average salary of 5 million, components of a total that do not add up to that total. Correction of such errors is often necessary to prevent flaws and inconsistencies in statistics to be published. The process of checking and correcting data is called data editing, see e.g. De Waal (2008), De Waal *et al.* (2011) and Pannekoek *et al.* (2013). Data editing consists of two main steps: error detection and error correction. The focus of this paper will be entirely on error detection. Traditionally, data editing is conducted manually, using subject matter knowledge. In this process returned questionnaires are checked individually and corrected if necessary. This approach is expensive and time-consuming. Approximately 25 to 40 per cent of the total survey budget was spent on data editing (Granquist and Kovar, 1997).

Today, data editing is more and more automated, using computer programs without user intervention. The basis of automatic editing is edit rules: formal relationships between variables. A simple example is that men cannot be pregnant. Usually, in business statistics, many edit rules are defined, that have many variables in common. In this way, connected systems of edit rules are obtained.

The ongoing automation of data editing means that more and more subject matter knowledge is translated into edit rules. Specification of edit rules is often done by subject matter experts, who repeatedly add new edits to the set of existing rules. Thus, the number of edit rules for one survey can grow to a very large number. A large number of edits may cause problems. Firstly, computational problems may arise: computation time can be long or – in the worst case – automatic editing of a record can be impossible. In this way, automated editing becomes the victim of its own success. Secondly, when the number of edits is large, it may become difficult to understand the joint effect of the defined rules. Undesirable, non-intended implications can occur, for example one rule that reverses the effects of another rule. Such problems can be avoided by removing erroneous and redundant rules and simplifying rules as much as possible. Erroneous edits are rules that have different implications than intended. For example, the following type of edit was found in an edit set, actually used by Statistics Netherlands:

If Questionaire_ID $\neq$ 1 OR Questionaire_ID $\neq$ 2 THEN VariableX = VariableY

Here, the OR-operator was meant to be an AND-operator. One can easily show that the conditional part of this edit is always satisfied. It may also happen that erroneous edits render an edit set infeasible, meaning that no combination of values satisfies all rules. In general, erroneous edits negatively affect the quality of automated edited data. As a result, manual correction of automated edited data may become necessary. Redundant edits are rules that are declaratively implied by other edits. Such edits can be removed from an edit set, without affecting results. The easiest example is a duplication of rules. Redundancy is often introduced if different persons specify edits at different moments of time, as nobody wants to bear the risk of omitting relevant rules (Chklovski and Gil, 2005). According to Sumathi and Paulraj (2013) the presence of redundant rules is a common situation in formulating large LP problems, a problem that is comparable to defining rules in data editing. It can be very difficult to identify redundant, erroneous and unnecessarily complicated rules by hand, especially if the number of edit rules is large. Fortunately, mathematical methods are available for this purpose. An advantage of using formal, mathematical methods is that users do not have to bother about redundancy and the complexity of edit rules; they can specify as much as rules as desired, simplification and error detection are done automatically out of sight.

Edits in data editing can be more formally specified as constraints on values. Therefore, the problem of edit rules simplification can be more generally stated as a problem of constraint simplification. A large number of methods for constraint simplification is available from the literature. Most algorithms are described in the context of mathematical optimization, for example: Paulraj and Sumathi (2010), Chinneck (1997) and Telgen (1983) and constraint satisfaction problems: Chmeiss *et al.* (2008) and Piette (2008). Applications of these algorithms can be found in a wide range of areas within operations research and artificial intelligence, amongst others: modeling customer's preferences in product customization (Felfernig *et al.*, 2014), testing under which conditions software can be applied (Dillig *et al.*, 2010) and soccer league scheduling (Bakker *et al.*, 1993). Remarkable is that algorithms for constraint simplification are not often mentioned in the context of data editing. However, few software applications with constraint simplification facilities are available. Statistics Canada developed the SAS application BANFF for economic surveys. This application includes certain methods for constraint simplification (see BANFF, 2005, Chapter 2), but does not allow for conditional (IF-THEN) edits that are frequently used in official statistics. At Statistics Netherlands the R package "Editrules" has been developed (Van der Loo and de Jonge, 2011, 2012) and De Jonge and Van der Loo (2014) and currently, a new package "Validate" is being developed. These packages include basic features for edit rules correction and simplification.

This paper contributes to fill the gap for constraint simplification techniques for data editing purposes. A new procedure will be proposed, based on a combination of algorithms that are already available from operations research and artificial intelligence. The procedure will be geared towards the type of edit rules that are frequently occurring in official statistics. Further, it will be demonstrated that these algorithms can actually be used to simplify real-life edit sets. The proposed procedure has been implemented in prototype of software that is planned to be incorporated in Editrules / Validate.

The structure of this paper is as follows. Section 2 describes functionalities of the new procedure. Section 3 presents formal, mathematical algorithms. Section 4 shows results of applications to real-life edit sets. Section 5 finishes this paper with a discussion.

**Acknowledgements**
The author is grateful for useful advice and suggestions from Edwin de Jonge, Mark van der Loo, Jeroen Pannekoek and Sander Scholtus.

# 2. Simplification features

In this section we first explain the type of edit rules that are frequently occurring in official statistics: unconditional and conditional linear (in)equalities. Then, a new procedure is described for edit rules simplification.

## 2.1 Edits

The purpose of this subsection is to explain the format of edit rules that are considered in the remainder of this paper.

### Unconditional edits

Examples of unconditional edits are:

Number of employees ≥ 0;

Total turnover = Domestic turnover + Foreign turnover.

In mathematical terms, unconditional edits are written as a system of linear equalities and / or inequalities. The general form is given by:

$$\boldsymbol{A}_E\boldsymbol{x} = \boldsymbol{b}_E,$$
$$\boldsymbol{A}_{I1}\boldsymbol{x} \le \boldsymbol{b}_E,$$
$$\boldsymbol{A}_{I2}\boldsymbol{x} \ge \boldsymbol{b}_E,$$

where $\boldsymbol{A}$ is a matrix of coefficients and $\boldsymbol{b}$ a vector of constants. The subscript *E* is used for equality constraints and *I1* and *I2* for inequality constraints.

### Simple Conditional edits

An example of a conditional edit rule is:

IF number of employees > 0 THEN wages > 0.

If the IF-statement is violated, a conditional ("IF-THEN") rule is always satisfied. Otherwise, if the IF-statement is fulfilled, an IF-THEN rule is satisfied, if the THEN-statement is fulfilled. From this, it follows that either the IF-statement needs to be violated, or the THEN-statement needs to be fulfilled. Thus, our previous rule can be rewritten as:

number of employees ≤ 0 OR wages > 0,

where the first component is the negation (i.e. opposite) of the "IF"-part and the second component expresses the "THEN"-part of the original edit. In general, simple conditional edits can be rewritten in this way.

## Compound conditional edits

Compound conditional edits are a generalization of simple conditional edits. Compound edits may contain:

- AND-statements in the IF-clause and/or;
- OR-statements in the THEN-clause.

An example of an edit in this form is:

IF number of employees > 0 AND turnover > 0 THEN
wages > 0 OR labour costs > 0.

It can be derived that compound statements of this form are satisfied, if one or more components in the IF-clause are violated, or if at least one of the components in the THEN-clause is fulfilled. Analogous to simple conditional edits, it follows that compound conditional edits can be rewritten as a combination of single statements, that are combined by OR-operators. For example, the above mentioned edit rule is equivalent to:

number of employees ≤ 0 OR turnover ≤ 0 OR wages > 0 OR labour costs > 0,

where the first two components are the opposite of the two IF-clause statements and the last two components expresses the THEN-clause of the original formulated edit.

More in general, simple and compound conditional edits can be reformulated in the so-called disjunctive normal form (DNF):

$C_1$ OR $C_2$ OR $C_3$ OR…

where $C_i$ denote single inequality and equality constraints, see for instance Hooker (2000) for an explanation of this concept.

Note that above we do not consider conditional edit rules, with:

- OR-operator in the IF-clause and/or
- AND-operators in the THEN-clause.

However, such edits can be rewritten as a number of simple and/or compound conditional edits that are in the above defined forms. For example, the edit:

IF number of employees > 0 OR turnover > 0 THEN
wages > 0 AND labour costs > 0

is equivalent to the combination of the following four “simple conditional rules”:

IF number of employees > 0 THEN wages > 0,
IF number of employees > 0 THEN labour costs > 0,
IF turnover > 0 THEN wages > 0,
IF turnover > 0 THEN labour costs > 0.

that can each be converted into disjunctive normal form.

In the remainder of this paper it will be assumed that unconditional edits are expressed as a system of linear (in)equalities and that conditional edits are written in disjunctive normal form.

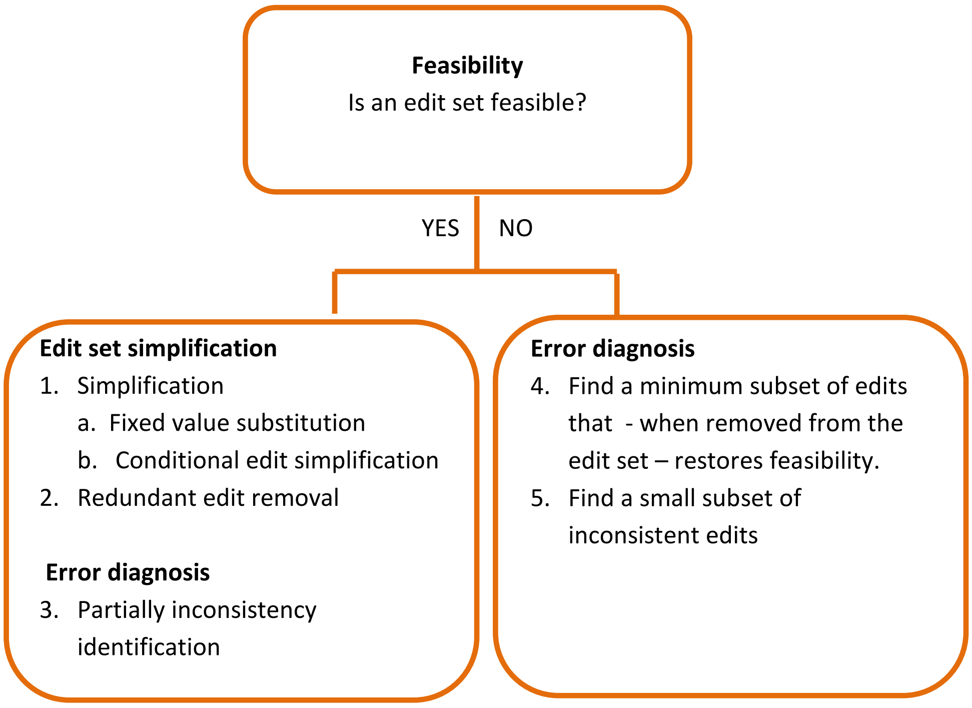
## 2.2 Procedure

First, we provide a schematic overview of features in the new procedure for simplifying constraints. Then, examples are given of each feature. A more formal description follows later in Section 3.
The first step in the overview of features in Figure 2.2.1 consists of a feasibility check of an edit set. An edit set is feasible if a set of values exists that satisfies all edit rules. A non-feasible edit set is also called contradictory or unsatisfiable. A nonfeasible edit set is given by:

Edit 1: $x > 10$,
Edit 2: $x = 5$.

For feasible edit sets, a simplification step and an error diagnosis step are applied. In the simplification step unnecessarily complicated edits are replaced by simpler rules. The error diagnosis step is aimed at detecting edits with unforeseen implications. For infeasible edit sets error diagnosis is applied with the purpose of localizing inconsistent rules.

### 2.2.1 Schematic overview of simplification features


**Feasibility**
Is an edit set feasible?
YES
NO
**Edit set simplification**
1. Simplification
a. Fixed value substitution
b. Conditional edit simplification
2. Redundant edit removal
**Error diagnosis**
3. Partially inconsistency identification
**Error diagnosis**
4. Find a minimum subset of edits that - when removed from the edit set – restores feasibility.
5. Find a small subset of inconsistent edits

### Simplification of feasible edit sets

In this subsection several methods for edit set simplification are described. The result of each step is a simplified edit set, with fewer variables or constraints than in the initial edit set.

### *Step 1a. Fixed value substitution*

It may happen that certain variables can attain only one value. Then, this value can be substituted in all edits in which it appears. An example is given below:

Edit 1: $x_1 + x_2 + x_3 = 10,$
Edit 2: $x_1 + x_2 \geq 10,$
Edit 3: $x_3 \geq 0.$

It is immediately clear that $x_3$ necessarily has to be zero; this variable is said to have a fixed value. After substitution of $x_3$ , the new rules are given by:

Edit 1': $x_1 + x_2 = 10,$
Edit 2': $x_1 + x_2 \geq 10,$
Edit 3': $0 \geq 0.$

Further a new edit needs to be defined to express the fixed value. In our example:

Edit 4: $x_3 = 0.$

Of course, these edits can be further simplified. Edit 3' ("0 ≥ 0)" is obviously 'always true' and can be removed from the edit set. Edit 2', is redundant, because it is implied by Edit 1'. The further removal of redundant edits is performed in a following step of the procedure (Step 2).

### *Step 1b. Conditional edit simplification*

Step 1b is designed for compound edits only. As mentioned, in previous subsection, such edits can always be written in the following form:

$C_1$ OR $C_2$ OR $C_3$ OR…

An example is:

$x_1 < 0$ OR $x_2 < 0$ OR $x_3 < 0$

Two substeps are distinguished below: removal of non-relaxing and non-constraining components.

### *Step 1b.I: Non-relaxing components*

A disjunction of edit statement may include components that can never be satisfied, given the edits in an edit set $E$. Such edit statements are called non-relaxing, see also Dillig et. al (2010). Non-relaxing components can be removed from a compound edit. An example:

Edit 1: $x_1 > 0$ OR $x_2 > 0$ OR $x_3 > 0,$
Edit 2: $x_2 < 0.$

Edits 2 implies that the statement $x_2 > 0$, within Edit 1, cannot possibly be satisfied, it is 'always FALSE'. Therefore it can be deleted from the edit set. We obtain:

Edit 1': $x_1 > 0$ OR $x_3 > 0.$

The number of terms in a compound edit statement diminishes by this kind of simplification.

In many cases, only one term remains. If this happens, a compound edit statement is transformed into a single edit statement. Consider the following example

Edit 1: $x_1 > 0$ OR $x_2 > 0$,
Edit 2: $x_2 < 0$.

Because " $x_2 < 0$" cannot occur, Edit 1 can be replaced by the 'single' edit statement:
Edit 1': $x_1 > 0$.

Replacing a compound edit by a single edit is very beneficial, because processing of single edits requires less computational effort than compound edits. Adding one conditional edit may lead to a doubling of computation time (Van der Loo and De Jonge, 2012).

### *Step 1b.II: Non-constraining components*

Compound edits may contain components that are – in combination with all other edits – always fulfilled. Such components are called non-constraining, see Dillig *et al.* (2010). In other words: these are 'always true'. Compound edits with a non-constraining component can be replaced by a single edit. Consider the following edits:

Edit 1: $x_1 < 50$ OR $x_2 > 100$,
Edit 2: $x_1 > 100$ OR $x_2 > 0$,

For all possible $x_1$ values, at least one of the statements "$x_1 < 50$" and "$x_1 > 100$" is violated. The two edits imply that either "$x_2 > 100$" or "$x_2 > 0$" necessarily has to be true. From this it follows that "$x_2 \leq 0$" cannot occur. In other words: "$x_2 > 0$" is a non-constraining component.

Because non-constraining components are always fulfilled, non-constraining components of a compound edit can be added to an edit set as if they were a single, unconditional edit. In our example, we can add "$x_2 > 0$" to the edit set.

After doing this, compound edits with a non-constraining component can be removed, because the added unconditional rules imply that these compound edits are always satisfied. In our example, Edit 2 can be omitted, after inserting "$x_2 > 0$" to the edit set, because Edit 2 will always be satisfied. As already mentioned above, it is always beneficial to remove compound edit statements from an edit set.
The resulting edit set is given by

Edit 1: $x_1 < 50$ OR $x_2 > 100$,
Edit 2': $x_2 > 0$

### *Step 2. Redundant edit removal*

Redundant edits are non-constraining edits. These edits can be left out of an edit set, without affecting results. Consider the following example:

Edit 1: $x_1 < 10$ ,
Edit 2: $x_1 < 12$.

As Edit 2 is implied by Edit 1, it does not reduce the ‘feasible set’ and it can be removed accordingly. Another simple example of a redundant edit is the edit “5 > 4”. Redundant edits may emerge as a result of constraint simplification (Step 1). Therefore it is important that Step 2 is performed after Step 1a and Step 1b.

## Error diagnosis for feasible edit sets

The aim of this subsection is to describe one method from literature that can be applied for error diagnosis of feasible edit sets.

### *Step 3. Partially inconsistency identification*

It may occur that an edit set is feasible as a whole, but infeasible for certain subdomains of variables. Bruni and Bianchi (2012) call this partially inconsistency. An example:

Tax $\geq 0.33$ Income ,

Tax $\leq 720$.

is a feasible edit set. However, the two edits become contradictory if income would be larger than 2160. Partial inconsistencies may involve multiple variables, e.g. an inconsistency may occur for "Wages > 2900" and "Length of career < 10".
Detection of partial inconsistencies is important, because it can be a way to detect errors. In our first example the rule “Tax ≤ 720” may be erroneous.
It is important to stress that deriving partial inconsistencies only gives a clue about wrongly formulated edits. In our first example, it can be correct that income cannot be larger than 2160. Expert knowledge is necessary for a final judgement about the correctness of an edit rule.

In data editing literature a distinction is made between explicit and implicit edit rules. Explicit rules are the user-defined rules of an edit set. Implicit rules can be derived by combining explicit rules. The notion “partially inconsistency” is equivalent to an implicit edit. As partial inconsistencies with one variable are the easiest to interpret, in Subsection 3.6 an algorithm will be proposed to identify these partially inconsistencies.

## Error diagnosis for infeasible edit sets

The error diagnosis step for infeasible edits consists of two substeps (Step 4 and 5 below). Both substeps are well-known in literature, see e.g. Mousseau *et al.* (2003).

### *Step 4. Find a small subset of edits that – when removed from the edit set – restores feasibility*

Contradictory edit sets are useless in practice. Therefore one may wish to transform an infeasible edit set into a feasible one. Infeasibility can be resolved by deleting edits from the original edit set. Assuming that formulation errors are made incidentally, in a non-systematic way, it is makes sense to delete as few edits as possible. Thus, in this step, we consider a method of transforming an infeasible edit set into a feasible one, by deleting as few edits as possible.

The edits that are designated to be deleted may be wrongly formulated edits, but this is not necessarily so. It is also possible that wrongly formulated edits are preserved in the feasible edit set, whereas correctly formulated edits are deleted. Below, under 5), a method will be presented to more directly locate erroneous edits.
The problem of finding a minimum subset to remove from an edit set can also be viewed as the problem of finding a maximum feasible subset from an infeasible edit set. The latter problem is also known as the maximum feasible subsystem problem (see e.g. Amaldi *et al.* 1999).
Consider the following example:

Edit 1: $x_1 = 4,$
Edit 2: $x_2 = 5,$
Edit 3: $x_3 = 4,$
Edit 4: $x_1 + x_2 = 10,$
Edit 5: $x_1 + x_2 + x_3 = 15.$

It can be seen that at least two edits need to be removed to restore feasibility, for example: Edit 1 and 3. If we choose to eliminate Edits 1 and 3 we obtain a feasible subset of edits. The same is true if we eliminate Edits 2 and 4.

### *Step 5. Find a small subset of inconsistent edits (smallest IIS)*

Above, a method is given to transform an infeasible edit set into a feasible one. Here, we will describe a more specific method to pinpoint a cause for inconsistency.
In general, it can be hard to find the cause of a contradiction, especially if the number of edit rules is large. For small edit sets, it is much easier to find out the reason for inconsistency.

Therefore, we will concentrate on isolating a smallest possible subset of inconsistent edit rules: a so-called an irreducible inconsistent subset (IIS). Within an IIS, consistency can be resolved by deleting one of its rules.
The idea is that erroneous rules are identified from an IIS. Subsequently, those rules are corrected or omitted. If it is not possible to resolve all inconsistencies in this way the procedure under 4) can be applied.

Inconsistent edit sets may contain several IIS´s. Even a single formulation error may result in multiple IIS´s, possibly with overlapping edits. To restore feasibility, at least one edit rule of each IIS needs to be deleted. Therefore, the problem mentioned under 4) can be formulated as a minimum-cardinality IIS set-covering problem. That is, the problem of finding a minimum number of edits that covers all IIS´s. Our purpose here is to identify one IIS. The procedure can be repeated to find multiple IIS’s. In particular, our interest will be on the smallest IIS´s, as this IIS most clearly points to a source of contradiction.

In our previous example Edits 1, 2 and 4 constitute an IIS. Two other IIS´s are given by the Edits 3, 4 and 5 and the Edits 1, 2, 3 and 5. The Edits 1, 2, 3 and 4 are contradictory, but those do not constitute an IIS, since that set can be further reduced. We are especially interested in the two IIS´s that contain three edits, as these are the smallest IIS´s.

# 3. Algorithms

In this section we describe the mathematical algorithms for each of the aforementioned steps. These algorithms have in common that they can be implemented by using Mixed Integer Programming (MIP) solvers. MIP-solvers are widely available; we used the R package lpSolveAPI (Konis, 2011). Before we describe the mathematical algorithms, we first explain the general form of a MIP problem.

## 3.1 Mixed Integer Programming

In general, MIP problems have the following form:

Minimize $f(\mathrm{x},\boldsymbol{z}) = \boldsymbol{c}^T \begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{pmatrix}$,

s.t. $\boldsymbol{R}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{pmatrix} \leq \boldsymbol{d}$,

where $\boldsymbol{c}$ is a constant vector ($\boldsymbol{c} \in \mathbb{R}^n$) and $\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{pmatrix}$ is a vector consisting of real ($\boldsymbol{x}$) and integer ($\boldsymbol{z}$) decision variables. One usually refers to $\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{pmatrix}$ as the decision vector and the inner product $\boldsymbol{c}^T \begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{pmatrix}$ as the objective function. Further, $\boldsymbol{R}$ is a coefficient matrix and $\boldsymbol{d}$ a vector of upper bounds (De Jonge and Van der Loo (2014)). As explained in De Jonge and Van der Loo (2014) all edit rules that are used in the R-package Editrules can be translated into constraints of a mixed integer problem. We will demonstrate this in Appendix A.

## 3.2 Determining feasibility

An edit set *E* is feasible if a set of values exists that satisfies all edits in *E*.
In practise, the feasibility of an edit set can be checked by applying a MIP-solver. This can be done by defining a MIP-programming problem, with $\boldsymbol{c} = 0$ and constraints that express the mathematical formulation of an edit set. The MIP-problem minimizes a constant objective function. Of course, if an optimum value exists, it will always be zero. In case of contradictory constraints, an error message will be returned.

## 3.3 Identifying fixed values

Fixed values are variables that can attain only one value, given the edits in an edit set *E*. Such values can be identified by defining and solving two MIP-programming problems for each real decision variable. In one of the problems the variable is maximised, in the other problem the variable is minimised. Both problems are under the constraints that all edits rules are satisfied.
If the minimum and maximum value turn out to be the same, the variable at hand can only attain one value. In that case an edit set can be simplified by adding a

constraint stating that a certain variable can only have one value and substituting this fixed value in all other edits in which it appears.

### Example

Consider the following edits:

Edit 1: $x_1 \leq 10$,
Edit 2: $x_1 \geq 10$.

To identify the fixed values, the following optimization problems are solved:

Min $x_1$
s.t. $x_1 \leq 10$,
$x_1 \geq 10$

and

Max $x_1$
s.t. $x_1 \leq 10$,
$x_1 \geq 10$

In this way, the minimum and maximum value are found to be 10. As $x_1$ can only attain one value; it is said to be a fixed variable. The value 10 is substituted in Edits 1 and 2. We obtain the new simplified constraints “10 ≤ 10” and “10 ≥ 10”. Further, a new constraint “$x_1$ =10” is added.

## 3.4 Conditional edit simplification

### Non-relaxing components

As explained in Subsection 2.2., a ‘non-relaxing’ component of a compound edit is a statement that cannot be satisfied, given the edits in an edit set *E.* A component which is always FALSE. This leads to the following definition:

*Definition*: A component $e_{ij}$ of a compound edit $e_i$ within an edit set *E* is non-relaxing if $E \cup e_{ij}$ is infeasible.

Here, $E \cup e_{ij}$ stands for the edit set that is obtained by extracting a compound edit’s component $e_{ij}$from $e_i$ and adding it to the set *E*, as if it were a single edit. The following algorithm is applied for identifying non-relaxing components:

**Algorithm 1**: Identification & removal of non-relaxing components

```
Input:    Edit set E
Output: Edit set E, without non-relaxing components.
1 For each compound edit $e_i \in E$ do
2   For each component $e_{ij} \in e_i$ do
3     $E^* \leftarrow E \cup e_{ij}$;
4     IF isFeasible($E^*$) = FALSE THEN $e_i \leftarrow e_i \setminus e_{ij}$
5   Next
6   IF NumberofComponents($e_i$)=1 THEN Compound edit $e_i$ needs to be
    considered as a single edit
7 Next
```

In each step of the algorithm one edit is added to an edit set *E*. Subsequently, the feasibility of the adapted edit set is checked. In Subsection 3.4.2 and 3.5 two other algorithms will be presented with the same structure as Algorithm 1.
This procedure can be implemented in the following way. For each component of a compound edit, a MIP programing problem is defined, with a constant objective function and constraints that express the mathematical translation of an edit set. This is similar to the algorithm for determining feasibility in Subsection 3.3. However, one additional constraint is defined that expresses the component from a compound edit. Then, it is verified whether a solution exists that satisfies all constraints. If such a solution does not exist, the compound edit's component is non-relaxing.
As mentioned before, non-relaxing components can be removed from a compound edit. If, after simplification of a compound edit statement, an edit is obtained that includes one component only, that edit is not a compound edit anymore, but a single edit.

### Example

Consider the following edits

Edit 1: $x_1 > 0$ OR $x_2 > 0$,
Edit 2: $x_2 < 0$.

First, we check whether the component "$x_1 > 0$" is non-relaxing. For this purpose, a MIP problem is defined. The constraints of the MIP are based on the following rules

$x_1 > 0$ OR $x_2 > 0$,
$x_2 < 0$,
$x_1 > 0$.

It can easily be seen that these constraints are feasible. Therefore, "$x_1 > 0$" is not a non-relaxing component. Next, it is checked whether "$x_2 > 0$" is a non-relaxing component. Again, a MIP-programming problem is defined, whose constraints are given by a mathematical representation of the following edits:

$x_1 > 0$ OR $x_2 > 0$,
$x_2 < 0$,
$x_2 > 0$.

It can easily be verified that these constraints are contradicting. From this it follows that "$x_2 > 0$" is a non-relaxing component. It cannot be satisfied, together with the edits in *E*. Hence, it can be removed from Edit 1. As a result we obtain the single edit: Edit 1' : $x_1 > 0$.

## Non-constraining components

A 'non-constraining' component of a compound edit is a statement for which the edits in an edit set *E* imply that it is 'always satisfied': a statement that is always TRUE. To identify non-constraining components, we make use of the following equivalence: the statement that a compound edit's component is always satisfied, is identical to the statement that the opposite of that component cannot occur. This leads to the following definition:

*Definition*: A component $e_{ij}$ of a compound edit $e_i$ within an edit set *E* is non-constraining if $E \cup \neg e_{ij}$ is infeasible. (where $\neg$ stands for negation)

The following procedure can be applied for identifying non-constraining components:

**Algorithm 2**: Identification of non-constraining components

```
Input:    Edit set E
Output: Edit set E, without non-constraining components.
1 FOR each compound edit $e_i \in$ E DO
2       FOR each component $e_{ij} \in e_i$ DO
3             $E^* \leftarrow E \cup \neg e_{ij}$ ;
4             IF isFeasible($E^*$) = FALSE THEN $E \leftarrow \{E \backslash e_i\} \cup e_{ij}$ END IF
5       NEXT
6 NEXT
```

As mentioned before, Algorithm 2 has the same structure as Algorithm 1; in each step of the algorithm a given edit set *E* is modified and the feasibility of the resulting edit set is checked.

This algorithm can be implemented as follows: for each component of a compound edit, a MIP programing problem is defined, without objective function and with constraints that express the mathematical translation of the edits from an edit set. Further, one additional constraint is defined that expresses the negation of a compound edit's component. Then, it is verified whether a solution exists that satisfies all constraints. If such a solution does not exist, the compound edit's component is non-constraining and the compound edit is replaced by the non-constraining component of that edit.

## Example

Consider, the following constraints:

Edit 1: $x_1 < 50$ OR $x_2 > 100$,
Edit 2: $x_1 > 100$ OR $x_2 > 0$.

Suppose we want to know whether "$x_2 > 0$" is non-constraining or not. According to the above mentioned procedure a MIP is defined, in which the constraints expresses the following constraints:

$x_1 < 50$ OR $x_2 > 100$ (Edit 1),
$x_1 > 100$ OR $x_2 > 0$ (Edit 2),
$x_2 \leq 0$ (the negation of "$x_2 > 0$").

It can be easily seen that these constraints are infeasible. Therefore, it follows that "$x_2 > 0$" is a non-constraining component. Because Edit 2 includes this non-constraining component, it will be replaced by the single edit "$x_2 > 0$".

## 3.5 Redundant constraint removal

In the literature a large number of methods has been mentioned for detection of redundant constraints. Paulraj and Sumathi (2010) performed a comparative study. We describe a method from Felfernig *et al.* (2011) and Chmeiss *et al.* (2008). The reason for choosing their method is its simplicity and the possibility of implementing the method by using a mixed integer programming solver.
First, we will present the definition of redundancy, then we will present the algorithm. An edit is redundant if all other edits imply that the edit is 'always satisfied'. This is equivalent to the statement that the opposite of the edit (in mathematical terms: negation) cannot occur. This leads to the following definition:

*Definition*: An edit $e_i$ from an edit set *E* is redundant, if $\{E \backslash e_i\} \cup \neg e_i$ is infeasible.

The edit set $\{E \backslash e_i\} \cup \neg e_i$ is obtained from *E*, by replacing Edit $e_i$ by its negation. The following pseudo-code can be used to remove redundant edits

**Algorithm 3:** Identification & removal of redundant edits

**Input:** Edit set *E*

**Output:** Edit set *E*, without redundant edits

```
1 FOR each Edit $e_i \in E$ DO
2     $E^* \leftarrow \{E \backslash e_i\} \cup \neg e_i$ ;
3     IF isFeasible($E^*$) = FALSE THEN $E \leftarrow E \backslash e_i$
4 NEXT
```

Again, this pseudo-code can be implemented using a MIP-solver. A MIP-programming problem is specified for each Edit $e_i$. In each problem the feasibility of $\{E \backslash e_i\} \cup \neg e_i$ is checked. A non-feasible result means that Edit $e_i$ is redundant. As noted before, Algorithm 3 has the same structure as Algorithms 1 and 2. In each step, an edit from an edit set is modified and the feasibility of the resulting edit set is checked.
It is important that redundant edits are deleted, before further identification of other redundant edits, otherwise one bears the risk of identifying too many redundant edits. For example, if one edits appears twice in an edit set, both edits are redundant, but after deletion of the first edit, the second edit may not be redundant anymore.

Further, the result of Algorithm 3 are order-dependent. We do not advice on the order of treatment of edits here, but leave this as a topic for further research.
A few words about the negation of compound edits: the negation of a compound edit consists of a number of single edits. For example, the negation of: "$x > 0$ or $y > 0$" is given by: "$x \leq 0$" and "$y \leq 0$", a combination of two single edits. Thus: adding the negation of a compound edit to an edit set basically means that a number of single edits are added to that edit set.

**Example**
Consider the following edits:

Edit 1: $x_1 \geq 100$,
Edit 2: $x_2 \geq 100$ OR $x_1 \geq 0$.

Suppose we want to know whether Edit 2 is redundant. As explained above, we need to replace Edit 2 with the negate of that edit, given by $x_2 < 100$ AND $x_1 < 0$, a combination of two single edits. Thus, we obtain the following edit set:

Edit 1: $x_1 \geq 100$,
Edit 2': $x_2 < 100$,
Edit 2": $x_1 < 0$.

It can be clearly seen that these edits are contradictory, from which it follows that Edit 2 is a redundant edit.

For large edit sets the aforementioned algorithm may not perform very well, as one MIP-needs to be solved for each edit.

## 3.6 Partially inconsistency localisation

As explained in Subsection 2.2. partially inconsistencies are certain subdomains of variables for which an edit set would be infeasible. In the current subsection an algorithm will be proposed to detect partially inconsistencies with one variable. Analogous to the identification of fixed variables, in Subsection 3.3, we propose an algorithm based on deriving lower and upper bound for each real decision variable. A finite lower or upper bound means that a partially inconsistency is found.

It should be noted however that not all partially inconsistencies can be found in this way. Partially inconsistencies in which the middle part of the domain is excluded will not be detected, e.g. for income between 1000 and 2000. Further, partially inconsistencies that depend on more than one variable are not detected either, e.g. the combination of "Age > 50" and "Income > 2200". Thus, the proposed algorithm does not guarantee that all potential problems in edit sets can be found. Further research may yield methods with a broader scope of applicability.

## 3.7 Minimum subset of edits to be removed

This section applies to infeasible edit sets. Our aim is to identify a smallest possible subset of edits whose removal restores feasibility. This can be done by defining and solving an appropriate MIP-programming model. The formulation of this model is well-known in literature, amongst others: Mousseau *et al.* (2003) and Kim & Ahn (1999). This MIP-programming problem can be stated in the following form:

Minimize $f(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p}) = \sum p_i$

s.t. $\boldsymbol{R}\binom{\boldsymbol{x}}{\boldsymbol{z}} \le \boldsymbol{d} + M\boldsymbol{p}$,

where $M$ is a large, positive number.

In this formulation binary 'penalty variables' $p_i$ are introduced for each edit. The value of $p_i$ is one if the corresponding edit is designated to be omitted; otherwise it is zero.
The objective function minimizes the number of omitted edits. The constraints ensure that all remaining, non-omitted edits are feasible. As mentioned in Subsection 2.1 all edits can be represented in the form: $\boldsymbol{R}\binom{\boldsymbol{x}}{\boldsymbol{z}} \le \boldsymbol{d}$. It will be explained below that equality type of edits can be transformed into two inequality constraints. We add $+ Mp_i$ to the right hand side of each inequality. For example: the constraint $x < 100$ is rewritten as $x < 100 + Mp_1$. It follows that a constraint is always satisfied if the penalty variable $p_i$ equalizes one. Otherwise, if $p_i$ is zero, the constraint necessarily needs to be obeyed. From this it follows that only non-omitted constraints have to be satisfied.

After solving the aforementioned optimization problem, we obtain the optimum value of the objective function. If this optimum objective function value is zero, all penalty variables are zero, meaning that all constraints can be obeyed, without making the problem infeasible. Since we apply the method on an infeasible edit set, the optimum value of the objective function necessarily has to be larger than zero. That is, at least one constraint will be identified as an 'omitted constraint'. The constraints for which the penalty variable equals one are the constraints that can be removed from the edit set, such that the resulting edit set becomes feasible. Below two additional remarks are presented, about the formulation of constraints for equality edits and for compound edits.

*Equality edits*: Equality type of edits can be translated into two "≤" type of constraints. For example: the edit $x = 4$ can be expressed as combination of: $x \le 4$ and $-x \le -4$. For both constraints, the same penalty term needs to be used, because both constraints refer to the same edit. The transformed constraints are: $x \le 4 + Mp_1$ and $-x \le -4 + Mp_1$.

*Compound edits*: As explained in Appendix A, one compound edit statements can be translated into a number of constraints in the MIP-problem format. For a compound edit statement with $n$ components, $n$ +1 constraints are defined. The first $n$ constraints express one component of the compound statement. The last constraint ensures that at least one of the n components needs to be satisfied. The term "$+Mp_i$" is only added to the latter constraint. For example: the compound edit statement

“$x \leq 0$” or “$y \leq 0$” can be rewritten in MIP-format as: “$x \leq 0 + Ml_1$” , “$y \leq 0 + Ml_2$” and “$l_1 + l_2 \leq 1 + Mp_1$where $l_1$, $l_2$ and $p_1$ are binary variables. It can be seen that the compound edit is always satisfied if $p_1$is one. In that case, it does not necessarily have to be true that either “$x \leq 0$” or “$y \leq 0$” needs to be satisfied. If $p_1$is zero, the compound edit necessarily has to be obeyed: at least one of its components has to be fulfilled.

**Example**

We consider an example, that was mentioned before in Section 2.

Edit 1: $x_1 = 4$,
Edit 2: $x_2 = 5$,
Edit 3: $x_3 = 4$,
Edit 4: $x_1 + x_2 = 10$,
Edit 5: $x_1 + x_2 + x_3 = 15$.

The following minimization problem is solved in order to find a subset of edits whose removal leads to a feasible set:

Min: $p_1 + p_2 + p_3 + p_4 + p_5$
Subject to:
$x_1 \leq 4 + Mp_1,$
$-x_1 \leq -4 + Mp_1,$
$x_2 \leq 5 + Mp_2,$
$-x_2 \leq -5 + Mp_2,$
$x_3 \leq 4 + Mp_3,$
$-x_3 \leq -4 + Mp_3,$
$x_1 + x_2 \leq 10 + Mp_4,$
$-x_1 - x_2 \leq -10 + Mp_4,$
$x_1 + x_2 + x_3 \leq 15 + Mp_5,$
$-x_1 - x_2 - x_3 \leq -15 + Mp_5.$
where $p_1$ , …, $p_5$ are binary variables and $M$ is a large number.

In an optimal solution we have that $p_1 = 1$, $p_2 = 0$, $p_3 = 1$, $p_4 = 0$ and $p_5 = 0$, meaning that removal of Edits 1 and 3 results in a feasible edit set. In general, it is possible that different optimal solutions exist. In that case, most MIP-solvers will only present one solution. An obstacle of the aforementioned algorithm is that computation problems may arise, because of the many binary variables that are defined. As a solution to this problem the heuristics described by Chinneck (1996 and 2001) can be used.

## 3.8 Identifying a small IIS

In this section we describe a method for identifying a small, infeasible, subset of edits, a so-called IIS. The method is obtained from Chinneck (1997), where several algorithms for detecting IIS’s are described. We choose to apply ‘Algorithm 3: the elastic filter’. This algorithm does not necessarily produce the smallest possible IIS.

But, as mentioned by Chinneck (1997), the algorithm is “a remarkably effective heuristic for finding small-cardinality IIS’s”.
The proposed algorithm consists of two steps. In the first step a subset of edits is selected that contains at least one IIS. In the second step, edits that are not involved with a small-cardinality IIS are deleted from this subset. The algorithm is summarised below:

**Algorithm 4:** Identification of an irreducible inconsistent set (IIS)

```
Input:    Edit set $E$
Output: Edit set $F$ that constitutes an IIS
Initialise: $F \leftarrow \emptyset$
Step 1
1   Find a subset of edits $G$ ($G \subseteq E$), whose removal from $E$, results in a feasible
    edit set. To do this, apply the MIP-programming approach, described in
    Subsection 3.6.
2   REPEAT
     a) Extend $F$ with all edits that are designated to be omitted in the previous MIP-
        solution  ($F \leftarrow F \cup G$)
     b) Solve another MIP, to find a new subset G that can be removed from Edit set
        $E$, to restore feasibility. Ensure that all previously selected edits, the edits in
        $F$, cannot be designated again.
     UNTIL an infeasible MIP-programming problem is obtained.
Step 2
1    FOR each Edit $e_i \in F$ DO
2       $F^* \leftarrow F \setminus e_i$
3       IF isFeasible($F^*$) = FALSE THEN $F \leftarrow F \setminus e_i$
4    NEXT
```

The first step repeatedly applies the algorithm that is mentioned in last subsection. The main idea is that – in order to restore the feasibility of an edit set $E$ - at least one edit of each IIS needs to be removed. In each step new edits are selected, that have not been selected before. If all edits of an IIS were selected in previous iterations, it will no longer be possible to obtain a solution for the MIP-programming problem. Because in each iteration at least one unique member of each IIS is selected, the maximum number of iterations cannot be higher than the cardinality of the smallest IIS.

As a result of the first step we obtain a set of edits that contains at least all edits from one IIS, but additionally, it may also contain edits from other IISs. The second step is needed to filter out all edits that belong to one IIS. The algorithm that is applied in Step 2 is called the “Deletion Filter” (Chinneck, 1997). It is based on the idea that – as an IIS the smallest possible set of contradictory constraints – deleting one of the edits from an IIS inevitability leads to a feasible edit set. Therefore, we can select the edits of a certain IIS from a larger edit set, by omitting all edits that – when deleted– do not leave behind a consistent set.

**Example**

We would like to select an IIS in the example of Subsection 2.6. In the first step we select a set of subset of edits that can be removed from the edit set to restore feasibility. As we have seen in previous subsection, such a subset is given by Edit 1 and Edit 3. Again, we search for a minimum set of edits that can be removed from an edit set to restore feasibility, but we ensure that Edits 1 and 3 cannot be selected again. This can be achieved in the MIP-formulation by enforcing that $p_1$ and $p_3$ equal zero, for example by adding the two constraints: "$p_1$ = 0" and "$p_3$ = 0", or by deleting the variable $p_1$ and $p_3$ from the model. An optimal solution of the second application of MIP-programming is given by Edit 2 and Edit 4, i.e. we obtain that $p_2$ and $p_4$ are one, all other $p_i$ are zero. For the third time, we try to find a minimum set of edits that can be omitted from the original edit set, such that it leaves behind a feasible edit set, but we ensure that Edits 1, 2, 3 and 4 cannot be selected again. This is not possible: the result would be an infeasible problem.
We continue to Step 2. For each of the edits, within the subset of Edits 1, 2, 3 and 4, it is checked whether it can be removed, such that it still leaves behind an infeasible set. It turns out that Edit 4 can be removed. After removal of Edit 4 we still obtain an infeasible set. This contradictory edit set cannot be further reduced. Thus, we obtain the final result: an IIS, given by Edits 1, 2 and 3.

Instead of applying Step 1 and Step 2, it is also possible to apply Step 2, without Step 1. In that case, an IIS will be found as well, but it may include more edits than when both steps are applied. In the example above, an IIS will be found consisting of three edits: Edits 3, 4 and 5, i.e. an IIS which is as large as in the example above. A reason for omitting Step 1 can be to prevent computational problems, that may arise when repeatedly applying the algorithm of Subsection 3.6.

# 4. Applications

The aim of this section is to apply constraints simplification methods on 'real-life' and 'fictitious' edit sets. By doing this we would like to show that the described methods are practically useful. The following data sets were used:

1. Wholesale: Real-life edit set used for the 2007 Production Statistics for wholesale in agricultural products and livestock, for businesses with 10 employed persons or more;
2. Health-care: Real-life edit set used for a survey among welfare and childcare institutions;
3. Finance of Enterprises: Edit set under development, meant to be used for a survey on financing of enterprises;
4. Bruni & Bianchi: Small, fictitious edit set derived from the Bruni and Bianchi (2012) paper, Section 4. See Appendix C for a list of edits.

Table 4.1.1 summarizes the main properties and results for these data sets.

### 4.1.1 Application to four edit sets

| | Wholesale | Health care | Finance of Enterprises | Bruni & Bianchi |
|---|---|---|---|---|
| **Original edits** | | | | |
| Number of edits | 118 | 196 | 339 | 10 |
| of which conditional: | 12 | 114 | 5 | 1 |
| Number of variables in edits | 89 | 75 | 343 | 3 |
| **Simplification** | | | | |
| Fixed values | 0 | 2 | 0 | 0 |
| Conditional replaced by unconditional | 1 | 7 | 0 | 0 |
| Redundant edits | 18 | 10 | 35 | 4 |
| **Cleaned edits** | | | | |
| Number of edits | 100 | 186 | 305 | 6 |
| of which conditional: | 11 | 104 | 5 | 1 |

First of all, all four edit sets are feasible. This is quite logical, if an edit set were contradictory, it would be useless for practical application.

Health care's edit set includes two fixed values. The occurrence of those fixed values is easily understood: for both fixed values there is an edit stating that a variable can only attain one value (like "$x = 0$"). Even if only few fixed values are identified, it is still beneficial to substitute fixed values in all edits in which they appear, as this simplifies the further handling of edits.

Further, several examples were found in which a conditional edit can be replaced by an unconditional edit: one case for 'wholesale' and seven cases for 'health care'.
In most cases, this can be easily understood. An example from the health care survey:

$A > 0$,
IF $A > 0$ THEN $B > 0$.

where $A$ and $B$ stand for a variable with a complicated name, not worthwhile to mention. It is quite obvious that the second edit can be replaced by "$B > 0$". The second edit is an unnecessarily complicated way of stating that $B$ has to be larger than zero. In other cases, it is more difficult to understand why a conditional edit can be replaced by an unconditional edit. Consider the following example, taken again from the health care edit set:

IF $A > B$ THEN $A \leq 0$,
$B \geq 0$

The first unconditional edit can be replaced by $A \leq B$. To explain this, suppose that $A > B$. Then, according to the first edit it follows: $A \leq 0$. But, it also follows that $B < 0$, because of the assumption $A > B$. This, however, contradicts with the second edit. For 'Wholesale' one difficult-to-understand case is found of a conditional edit that can be replaced by an unconditional edit. As explained in Appendix B, this follows from the joint effect of nine edits.

In general, an important merit of automated procedures is that unnecessarily complicated constraints can be identified that would be very hard to detect by hand. Redundant edits were detected in all data sets. For “Health care” redundant conditional edits were found, that are implied from other conditional edits. An illustrative example is:

Edit 1: IF $A \geq 10B$ THEN $C \leq 0$,
Edit 2: IF $C < 1$ THEN $B \leq 0$,
Edit 3: IF $A \geq 10B$ THEN $B \leq 0$.

Here, Edit 3 is implied by Edit 1 and 2. It can be deleted, accordingly.
A frequently occurring case is redundancy of lower and upper bounds. A typical example is when one variable is a sum of other variables and each variable, including the sum, has to be at least zero. In that case, the lower bound for the sum is redundant. Nevertheless, users may find it informative to specify bounds for all variables. A great advantage of automated procedures, is that removal of redundant constraints can be done out of sight, so that users can still specify all possible bounds, without ending up with an inefficient edit set.

The result of the proposed algorithm for identifying redundant constraints is a set of redundant constraints. It may however be difficult to understand the underlying reason for redundancy. The same holds true for ‘redundant’ components of compound edits. The following solution can be applied: first, add the negate of a redundant constraint to an edit set. This will render an edit set infeasible. Next, identify an IIS. That IIS will contain the redundant rule, as well as a number of other edits that explain the redundancy. By applying this solution, it was determined for ´wholesale´ that 9 edits are involved with a redundant component of a compound edit (see Appendix B).

# 5. Discussion

From software verification to soccer league planning; a great variety of automated constraint handling problems is described in literature. An enormous number of constraints may be handled in those applications. Many authors have indicated that performance of these applications can be improved by conducting a constraint simplification step. However, to the best knowledge of the author, constraint simplification is not often applied in the field of data editing.

This paper shows that automated data editing can benefit from the available techniques for constraint simplification. A new procedure was developed, that combines several known algorithms from operations research and artificial intelligence. It is shown that real-life edit sets can actually be simplified by adopting the newly developed procedure.

Therefore, constraint simplification has the potential of improving computational performance of automated data editing. Consequently, the application possibilities of automated data editing can be further extended. An important practical merit of the proposed procedure is that simplification can be automated, out of sight of users, so that practitioners in the field do not have to bother about specifying constraints in a compact way.

Further, we have presented several methods for detection of erroneous constraints. In the worst case an erroneous constraints can render an edit set infeasible, for example if an erroneous rule contradicts with an other rule. By correcting wrongly specified rules, quality of automated edited data can be improved, reducing the need for manual intervention.

A direction for further research is to extend the application of constraint simplification techniques to the data editing problem. In this paper we considered numerical edits only. Application to categorical edits is a topic for additional research. Other topics for further research are: the order of treatment of edits within the algorithm for removing redundant edits and further development of algorithms for detecting partially inconsistency. Moreover, more algorithms may be available from operations research and artificial intelligence that are useful for data editing that are not mentioned in this paper. We end this discussion with three practical issues that have not mentioned before:

### Soft constraints
In many practical application a distinction is made between soft and hard constraints. Soft constraints need to be satisfied for the majority of cases, but, under certain circumstances, soft constraints may be violated. Because soft constraints do not necessarily have to be obeyed, it does not make sense to combine soft rules according to the rules of logic, in the way this is possible for hard constraints. For this reason the methods that are described in this paper are not meant to be used for soft constraints.

### Stratification variables
In many practical applications stratification variables are incorporated into the edits. For example: if “Industry code” = “*X*” THEN “ ….”. In this case, Industry code, is a stratification variable, which is not subject to change in the data editing process. Therefore, stratification variables do not have to be taken into account in a data editing process. It is better to apply independent edit sets for each stratum.

### Splitting edit sets
This papers implicitly assumed that edits are interconnected. However, if this is not the case, it is advisable to split an edit set *E* into disjunct sets, $\oplus_i E_i$, such that $e_i \in E_i$ and $e_j \in E_j$ $(i \neq j)$ do not have any variable in common. Disjunct edit sets can be treated independently, which may improve performance of data editing algorithms.

# 6. References

Amaldi, E., M. Pfetsch, L. Trotter Jr. (1999). Some structural and algorithmic properties of the maximum feasible subsystem problem, *Proceedings of the Integer Programming and Combinatorial Optimization conference* (IPCO'99), *Lecture Notes in Computer Science 1610*, Springer–Verlag, New York, 45–59.

Bakker R., F. Dikker, F. Tempelman & P. Wogmim (1993). Diagnosing and solving over-determined constraint satisfaction problems. In: *13th International Joint Conference on Artificial Intelligence*, 276–281, Chambery, France. http://ijcai.org/Past%20Proceedings/IJCAI-93-VOL1/PDF/039.pdf

Banff. (2005), *Functional Description of the BANFF System for Edit and Imputation*, Statistics Canada, internal document.

Bruni R. & G. Bianchi (2012). A Formal Procedure for Finding Contradictions into a Set of Rules. *Applied Mathematical Sciences*, 6, 6253 - 6271.

Chinneck J. W. (1996). An effective polynomial-time heuristic for the minimum-cardinality IIS set-covering problem, *Annals of Mathematics and Artificial Intelligence*, 17, 127‑144.

Chinneck J. W. (2001). Fast Heuristics for the Maximum Feasible Subsystem Problem, *INFORMS Journal on Computing*, 13, 210–223.

Chinneck J. W. (1997). Finding a useful subset of constraints for analysis in an infeasible linear program. *INFORMS Journal on Computing*, 9, 164–174. http://www.sce.carleton.ca/faculty/chinneck/docs/UsefulSubset.pdf

Chklovski T. & Y. Gil. (2005). An analysis of knowledge collected from volunteer contributors. In *20th National Conference on Artificial Intelligence* (AAAI-05), Pittsburg,PA, 564–571. http://www.coli.uni-saarland.de/courses/WebAsCorpus-12/papers/Learner.pdf

Chmeiss, A., V. Krawczyk & L. Sais. (2008). Redundancy in CSPs. In: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008), IOS Press, Amsterdam.

De Waal T. de (2008). *An overview of statistical data editing*, Discussion Paper 08018, Statistics Netherlands. http://www.cbs.nl/NR/rdonlyres/693E4B18-9322-4AC2-99FD-DB61F03637B2/0/200818x10pub.pdf

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

Dillig I., T. Dillig & A. Aiken (2010). Small formulas for large programs: On-line constraint simplification in scalable static analysis. In: Cousot R. and M. Martel, (eds), *SAS 2010,* LNCS 6337, 236–252. Springer. http://theory.stanford.edu/~aiken/publications/papers/sas10.pdf

Felfernig, A., Zehentner, C. & P. Blazek (2011). CoreDiag: Eliminating redundancy in constraint sets. In: *22nd International Workshop on Principles of Diagnosis*, Munich http://www.ist.tugraz.at/felfernig/images/stories/home/dx_corediag.pdf

Felfernig A., L. Hotz, C. Bagley & J. Tiihonen (2014). *Knowledged-based configuration: from research to business cases.* Morgan Kaufmann Publishers Inc., San Francisco.

Granquist, L. & 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., New York, 415-435.

Hooker J. (2000). *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction.* John Wiley & Sons, New York.

Jonge E. de & M. Van der Loo (2014). *Error Localization as a mixed integer problem with the Editrules package*, Discussion paper 201407, Statistics Netherlands. http://www.cbs.nl/NR/rdonlyres/A2940A3A-17DE-4A57-96EF-6292F6756319/0/201407x10pub.pdf

Kim S.H.& B.S. Ahn (1999), Interactive group decision making procedure under incomplete information, *European Journal of Operational Research*, 116, 3, 498-507.

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

Loo, M. van der & E. De Jonge (2011). *Manipulation of linear edits and error localization with the Editrules package*, Discussion Paper 201120, Statistics Netherlands. http://www.cbs.nl/NR/rdonlyres/21EDCB84-5891-4DB8-A826-F0FD9EB061E4/0/2011x1020.pdf

Loo, M. van der & E. De Jonge (2012). *Manipulation of conditional restrictions and error localization with the Editrules package*. Technical Report 201215, Statistics Netherlands. http://www.cbs.nl/NR/rdonlyres/3D14620B-51F8-4BCE-A404-AC9A2CC8608A/0/201215x10pub.pdf

Mousseau V., J.R. Figueira, L. Dias, C.G. Da Silva & J. Climaco (2003). Resolving inconsistencies among constraints on the parameters of an MCDA model. *European Journal of Operational Research*, 147, 72-93.

Pannekoek, J., S. Scholtus & M. van der Loo (2013), Automated and manual data editing: a view on process design and methodology, *Journal of Official Statistics*, 29, 4, 511–537.

Paulraj S. & P. Sumathi (2010). A Comparative Study of Redundant Constraints Identification Methods in Linear Programming Problems. *Mathematical Problems in Engineering*, 2010, Article ID 723402.

Piette,C. (2008). Let the solver deal with redundancy. In: *20th IEEE InternationalConference on Toolswith Artificial Intelligence* (ICTAI'08), Dayton, Ohio, 67–73.

Sumathi P. & S. Paulraj (2013). Identification of Redundant Constraints in Large Scale Linear Programming Problems with Minimal Computational Effort. *Applied Mathematical Sciences*, 7, 3963 - 3974.

Telgen J. (1983). Identifying Redundant Constraints and Implicit Equalities in Systems of Linear Constraints. *Management Science*, 29, 1209-1222.

# Appendix A. Edit rules as MIP-constraints

As explained in Section 3.1 a mixed integer programming (MIP) problem has the following general form:

$$\text{Minimize } f(\mathbf{x},\boldsymbol{z}) = \boldsymbol{c}^T \begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{pmatrix},$$

$$\text{s.t. } \boldsymbol{R}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{pmatrix} \leq \boldsymbol{d},$$

where **x** and ***z*** are vectors of real and integer decision variables, see Section 3.1 for an explanation. Below, we will demonstrate that all single and compound edits can be translated into constraints of a MIP-problem.

## Single edits

We will show that all kinds of 'single' edits with "≤", "<", "=", "≥" or ">" operators can be transformed into a "≤" type of constraint. An equality edit can be expressed as a combination of a "≤" and a "≥" constraint. For example: $x = 6$ is equivalent to $x \leq 6$ and $x \geq 6$. A "≥"-type of constraint can be transformed into "≤" constraint by multiplication by -1. For example, $x \geq 6$ is equivalent to $- x \leq 6$. In the same way, ">" constraints can be transformed into a "<" constraint. Finally, a "<" type of constraint can be transformed into a "≤" type of constraint by subtracting a small value from the right-hand side of the constraint. For example: $x < 6$ is equivalent to $x \leq 6 - \varepsilon$, where $\varepsilon$ is a sufficiently small number.

## Compound edits

A compound edit statement can be written in the following form:

$$\cup_{i=1}^{n} \boldsymbol{c}_i^T \boldsymbol{x} \leq d_i$$

Statement of this kind can be reformulated as a combination of $n + 1$ constraints, that fits the MIP-formulation for constraints. This reformulation is as follows:

$$\boldsymbol{c}_i^T \boldsymbol{x} \leq d_i + M z_i \quad \text{for } i = 1,..,n$$
$$\sum_{i=1}^{n} z_i \leq n - 1$$

where $z_i$ is an integer variable (≥ 0) and $M$ is a sufficiently large constant.

In this formulation, if $z_i$ is zero, the *i*th component of the compound statement has to be satisfied. If $z_i$ is larger than zero, the corresponding constraint will always be 'true', which implies that the *i*th component of the compound statement does not necessarily have to be obeyed. The last constraint ensures that at least one of the $z_i's$ will be zero; meaning that at least one component of a compound edit statement needs to be fulfilled.

**Example**

The compound statement: " x ≤ 10 or y ≤ 10" is modelled by the following three constraints:

$x \quad \leq 10 + Mz_1$

$y \quad \leq 10 + Mz_2$

$z_1 + z_2 \quad \leq 1$

where $z_1$ and $z_2$ are discrete variables and $M$ is a sufficiently large constant.

# Appendix B. A specific redundant edit

We will show an example from the wholesale edit set (see Section 4), in which a conditional edit can be replaced by an unconditional edit. This example is however relatively difficult to understand. The redundancy follows from the combined effect of nine edits. In the explanation below we will use the original edits, with the original variable names. The unnecessary conditional edit is given by:

```
IF PERSONS122000 > 0 THEN PERSONS110100 <=
PERSONS110000 + PERSONS122000
```

We will proof below that the "THEN" clause:

```
PERSONS110100 ≤ PERSONS110000 + PERSONS122000
```

is always satisfied. The nine edits, that explain this, are given by:

```
Edit 1: IF PERSONS122000 > 0 THEN PERSONS110100 ≤
    PERSONS110000 + PERSONS122000,
Edit 2: IF  PERSONS100000 > SUBTOWP100000 AND
    PERSONS122000 =  0  THEN PERSONS110100 <
    PERSONS110000,
Edit 3: PERSONS100000 = PERSONS111000 -
    PERSONS122000 + PERSONS130000 + PERSONS121000 +
    PERSONS113000,
Edit 4:SUBTOWP100000 = PERSONS111000 -
    PERSONS122000,
Edit 5: PERSONS130000 ≥ 0,
Edit 6: PERSONS121000 ≥ 0,
Edit 7: PERSONS113000 ≥ 0,
Edit 8: IF  PERSONS111000 = PERSONS100000 THEN
    PERSONS110100 =   PERSONS110000,
Edit 9: PERSONS122000 ≥ 0.
```

**Proof**

We consider four situations.

*Case 1: PERSONS122000 > 0*

The statement follows from Edit 1.

*Case 2: PERSONS122000 = 0 and PERSONS100000 > SUBTOWP100000*
The statement follows from Edit 2 and the assumption PERSONS122000=0.

*Case 3: PERSONS122000 = 0 and PERSONS100000 ≤ SUBTOWP100000*
From Edits 3-7, it follows that PERSONS100000 ≥ SUBTOWP100000. Combining this with our assumption PERSONS100000 ≤ SUBTOWP100000, yields PERSONS100000 = SUBTOWP100000. Because of Edit 4 and our assumption PERSONS122000 = 0, it follows that PERSONS100000 = PERSONS111000. Combining this with Edit 8 and and our assumption PERSONS122000 = 0, we finally arrive at PERSONS110100 ≤ PERSONS110000 + PERSONS122000.

*Case 4: PERSONS122000 < 0*
Not possible, because of Edit 9.

# Appendix C. Bruni and Bianchi edit set

Below, we present the edit set, called "Bruni and Bianchi" in Section 4. This edit set is a subset of the edits from the Bruni and Bianchi (2012) paper.

Edit 1 : Income ≤ 1000 + 20*Length_of_Career
Edit 2 : Income ≥ 200 + 30*Length_of_Career
Edit 3 : Tax ≥ 0.33 Income
Edit 4 : Tax ≤ Income
Edit 5 : IF Income ≤ 200 THEN Tax ≤ 100
Edit 6 : Length_of_Career ≤ 92
Edit 7 : Income ≥ 0
Edit 8 : Income ≤ 5000
Edit 9 : Tax ≥ 0
Edit 10:Tax ≤ 720

It can be derived that Edits 6, 7, 8 and 9 are redundant.
Short explanation: from Edit 1 and 2 it follows that "Length_of_Career ≤ 80", which makes Edit 6 redundant; from Edits 3 and 10 it follows that "Income ≤ 2160", Therefore Edit 8 is redundant; Edits 3 and 4 imply that that "Tax ≥ 0" and therefore Edit 9 is redundant. Combining this with Edit 4, yields that Edit 7 is redundant as well.

## Explanation of symbols

| | |
|---|---|
| Empty cell | Figure not applicable |
| . | Figure is unknown, insufficiently reliable or confidential |
| * | Provisional figure |
| ** | Revised provisional figure |
| 2014–2015 | 2014 to 2015 inclusive |
| 2014/2015 | Average for 2014 to 2015 inclusive |
| 2014/’15 | Crop year, financial year, school year, etc., beginning in 2014 and ending in 2015 |
| 2012/’13–2014/’15 | Crop year, financial year, etc., 2012/’13 to 2014/’15 inclusive |

Due to rounding, some totals may not correspond to the sum of the separate figures.

## Colofon

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

*Prepress*
Statistics Netherlands, Studio BCO

*Design*
Edenspiekermann

*Information*
Telephone +31 88 570 70 70, fax +31 70 337 59 94
Via contactform: www.cbsl.nl/information