DE GRUYTER OPEN

Journal of Official Statistics, Vol. 34, No. 1, 2018, pp. 27–39, http://dx.doi.org/10.1515/JOS-2018-0002

Constraint Simplification for Data Editing of Numerical Variables

Jacco Daalmans[1]

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 article shows that data editing can benefit from automated constraint simplification techniques. Performance can be improved, which broadens the scope of applicability of automatic data editing. Flaws in edit rule formulation may be detected, which improves the quality of automatic edited data.

Key words: Data editing; constraint simplification; conditional constraints; optimization.

1. Introduction

Collected micro data usually contain errors, for example, pregnant men, average salary of five million euro and 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 is called data editing, see, for example, De Waal et al. (2011) and Pannekoek et al. (2013). A common approach for data editing is based on the paradigm of Fellegi and Holt (1976), stating that the data in a record should be adapted to satisfy all edit rules by changing the fewest possible values.

Error localization according to the Fellegi and Holt paradigm can be formulated as a Mixed Integer Linear Program (MILP) problem, see, for example, De Waal et al. (2011). Although, in general, a solution to a data editing problem can be found in reasonable time – typically a few seconds – the worst-case performance of a MILP problem is known to be exponential in the problem size. Even when using modern computers, it can take hours or even days to obtain a solution for a single record.

From Operations Research and Artificial Intelligence it is well known that performance of a mathematical optimization problem can be improved by a constraint simplification step, see, for example, Paulraj and Sumathi (2010), Telgen (1983), Chmeiss et al. (2008), and Piette (2008). This means eliminating redundant constraints and simplifying unnecessary complicated constraints, before optimization. Nevertheless, remarkably few applications of constraint simplification are known in the context of data editing. Bruni (2005) explains how redundant edit rules can be detected. Also, Statistics Canada developed a software tool with

[1] Statistics Netherlands, PO Box 24500, 2490 HA Den Haag, The Netherlands. Email: j.daalmans@cbs.nl
Acknowledgments: The authors would like to thank three reviewers and the associate editor for useful comments that greatly contributed to improving the article. The author is also grateful for useful advice from Edwin de Jonge, Mark van der Loo, Jeroen Pannekoek, Sander Scholtus, and Ton de Waal.

simplification features (BANFF support team 2008, Chap. 2). These applications do however not allow for conditional ("IF-THEN") rules, where the variables involved in the IF and THEN statements may contain errors. Such rules frequently occur in official statistics and are especially problematic for computational performance, due to the integer variables that arise in the corresponding MILP problem.

This article contributes to fill the gap for constraint simplification techniques for error localisation of numerical data. Special attention will be given to conditional rules. We present automated methods that work at the formal level through solving MILP problems.

An advantage of automated procedures is that removal of redundant constraints can be done out of sight, so that users can still specify all possible rules without ending up with an inefficient edit set. Working at the formal level means that the methods can be applied to a generic class of rules, regardless of their semantic meaning.

Since edit rule simplification improves computational performance, it has the potential of further extending the application possibilities of automated data editing. Besides this, expert's feedback on automatically detected redundant edit rules might help to increase the understanding of the joint effects of a set of rules. Due to the complex interdependence and misspecification, a set of rules may have different implications than intended. Correction of erroneous rules improves the quality of automatic edited data and avoids the need for manual adjustment of results. For example, the following redundant rule was found in an edit set, actually used by Statistics Netherlands:

$$\text{IF}(\text{Questionaire\_ID} \neq 1 \text{ OR } \text{Questionaire\_ID} \neq 2)\text{THEN}(\text{VariableX} = \text{VariableY}) \quad (1)$$

Manual inspection might reveal that the OR-operator was meant to be an AND-operator.

The structure of this article is as follows. Section 2 describes the MILP formulation for data editing of numerical variables. Sections 3–5 present formal, mathematical algorithms for simplifying edit sets: eliminating single variables is described in Section 3, eliminating redundant parts from conditional rules is discussed in Section 4 and the redundancy of rules as a whole is considered in Section 5. Section 6 presents real-life applications of constraint simplification and data editing. Finally, Section 7 finishes this article with a discussion.

2. Outline of Basic Approach

We introduce the basic idea of MILP problems first. Then, it is explained how edit rules can be translated into MILP constraints.

2.1. *Definition of a MILP Problem*

A MILP problem consists of a loss function to be minimized and a set of inequality constraints involving both real and integer variables. A general form is given by

$$\begin{aligned} &\text{Minimize } f(\mathbf{x},\mathbf{z}) = \mathbf{c}^{\mathrm{T}} \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix}, \\ &\text{s.t. } \mathbf{A} \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix} \leq \mathbf{b}, \\ &\mathbf{x} \in \mathbb{R}^p \text{ and } \mathbf{z} \in \mathbb{Z}^q \end{aligned} \quad (2)$$

where $\mathbf{x}$ and $\mathbf{z}$ are vectors of real and integer decision variables, $\mathbf{c}$ is a constant vector ($\mathbf{c} \in \mathbb{R}^{p+q}$), $\mathbf{A}$ is a coefficient matrix and $\mathbf{b}$ a vector of upper bounds, see, for example, Bertsimas and Tsitsiklis (1997).

In the remainder of this article several algorithms are proposed that make use of the feasibility of a set constraints. This can be checked by a MILP solver by using a trivial loss function with $\mathbf{c} = \mathbf{0}$. Of course, if a solution exists the optimum value will be zero, but if the set of constraints is infeasible, most MILP solvers return an error message.

2.2. *Edit Rules as MILP Constraints*

This subsection introduces the edit rules that are considered in this article and explains how these rules can be transformed into MILP constraints. The edit rules in this article can be subdivided into unconditional and conditional rules.

We consider linear unconditional rules, like

$$\begin{aligned} &\text{Total turnover} = \text{Domestic turnover} + \text{Foreign Turnover},\\ &\text{Total turnover} \geq 0, \end{aligned}$$

that can be straightforwardly formulated as MILP contraints. One could note that the constraints in (2) do not allow for "larger than" and "equality" signs, but it is well-known that these rules can be reformulated into the required form. For example, an equality can be written as two inequalities and a constraint $x > 0$ can be approximated by $-\mathrm{x} \leq -\epsilon$, where ϵ is a sufficiently small value.

We also consider 'simple' and ' compound' conditional rules. A 'simple' conditional edit has the following form

$$\text{IF} < \text{Statement 1} > \text{THEN} < \text{Statement 2} >,$$

where each "statement" is a linear equality or inequality. Compound rules may also contain:

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

An example is:

$$\begin{aligned} &\text{IF}\,(\text{Number of employees} > 0 \text{ AND Turnover} > 0)\ \text{THEN}\\ &\quad(\text{Wages} > 0 \text{ OR Labour costs} > 0). \end{aligned} \tag{3}$$

Note that above we did not consider rules with:

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

but these rules can be rewritten as a number of simple conditional rules. For example, the edit:

$$\begin{aligned} &\text{IF}\,(\text{Number of employees} > 0 \text{ OR Turnover} > 0)\ \text{THEN}\\ &\quad(\text{Wages} > 0 \text{ AND Labour costs} > 0) \end{aligned}$$

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

$$\begin{aligned}
&\text{IF Number of employees} > 0 \text{ THEN Wages} > 0,\\
&\text{IF Number of employees} > 0 \text{ THEN Labour costs} > 0,\\
&\text{IF Turnover} > 0 \text{ THEN Wages} > 0,\\
&\text{IF Turnover} > 0 \text{ THEN Labour costs} > 0.
\end{aligned}$$

As mentioned by Chen et al. (2010), conditional rules need to be expressed in Disjunctive Normal Form (DNF), before these can be further converted into the required MILP format. A DNF is a disjunction of assignments (a sequence of OR's) that makes a rule *True*, see, for example, Hooker (2000).

To explain the transformation to DNF, note that a conditional rule is satisfied, if either the IF-clause is violated, or if the THEN-clause is fulfilled. Thus, a condition rule can be put in DNF, by joining the negation (i.e., opposite) of the "IF"-clause with the original "THEN"-clause. For example, the rule: "If Turnover $>$ 0 THEN Wages $>$ 0" can be stated as "Turnover $\leq$ 0 OR Wages $>$ 0".

For compound edits, the IF-clause is assumed to be a conjunction (sequence of AND's). According to Morgan's law, the negation of a conjunction is a disjunction of negations. To illustrate this, the example in (3) can be written in DNF as

$$\begin{aligned}
&\text{Number of employees} \leq 0 \text{ OR Turnover} \leq 0 \text{ OR}\\
&\text{Wages} > 0 \text{ OR Labour costs} > 0,
\end{aligned}$$

where the first two statements are negations of the original IF-clause statements.

An expression for n_C edit rules in DNF is given by

$$\bigcup_{\mathrm{j}=1}^{D_i} \left(\left(\boldsymbol{a}_{ij}^C \right)^T \mathbf{x} \leq b_{ij}^C \right) \quad i = 1, \ldots, n_C. \tag{4}$$

where an edit rule i is stated as a disjunction with D_i disjunctive terms. The coefficients and upper bounds for the jth term are denoted by $\boldsymbol{a}_{ij}^C$ and b_{ij}^C respectively. Again, 'equality', 'larger than' or 'smaller than' constraints can be reformulated into the form (4).

To express the constraints in (4) as MILP constraints, the following formulation can be used, based on the so-called Big M method.

$$\begin{aligned}
&\left(\boldsymbol{a}_{ij}^C \right)^T \mathbf{x} \leq b_{ij}^C + M(1 - z_{ij}), \quad i = 1, \ldots, n_C, \quad j = 1, \ldots, D_i,\\
&\sum_{j=1}^{D_i} z_{ij} = 1 \quad i = 1, \ldots, n_C,\\
&-z_{ij} \leq 0 \quad i = 1, \ldots, n_C, \quad j = 1, \ldots, D_i.
\end{aligned} \tag{5}$$

where z_{ij} are integer variables and M is a sufficiently large constant.

The equation $\sum_{j=1}^{D_i} z_{ij} = 1$ guarantees that only one disjunctive term is selected per disjunction. For each selected term (i, j with $z_{ij} = 1$), it is enforced that $\left(\boldsymbol{a}_{ij}^C \right)^T \mathbf{x} \leq b_{ij}^C$. For each non-selected term (i, j with $z_{ij} = 0$), the first constraint in (5) becomes redundant.

As shown in (5) integer variables are needed for the formulation of conditional rules. Because integer variables are much less efficiently handled than continuous variables, conditional rules can be less efficiently processed than unconditional ones. Therefore, it is very beneficial to replace conditional rules by unconditional ones.

3. Fixed Value Elimination

The aim of this technique is to shorten edit rules by elimination of 'fixed' variables, that is, variables with only one admissible value. As a result, an edit set may become simpler, possibly giving rise to a better performance of data editing software. Moreover, misspecification of edit rules might be detected by manual inspection of fixed values. Consider the following example:

$$\begin{aligned} &\text{Edit } 1\text{: } x_1 + x_2 + x_3 = 10, \\ &\text{Edit } 2\text{: } x_1 + x_2 \geq 10, \\ &\text{Edit } 3\text{: } x_3 \geq 0. \end{aligned} \tag{6}$$

It is immediately clear that x_3 necessarily has to be zero. In other words, x_3 is a fixed variable. Fixed values can be identified by solving two MILP programming problems for each continuous variable. Each variable is minimized and maximized once, subject to the MILP representation of the edit rules. If the minimum and maximum value turn out to be the same, the variable at hand is a fixed variable. Its value can be substituted in all edits in which it appears and a constraint is added stating that the fixed variable can only attain the fixed value.

Besides fixed values, any finite minimum or maximum is a candidate for content-wise analysis, because these bounds may be different than intended.

In our example, we can add the rule $x_3 = 0$ to our edit set and substitute x_3 in all other rules. We obtain

$$\begin{aligned} &\text{Edit } 1'\text{: } x_1 + x_2 = 10, \\ &\text{Edit } 2'\text{: } x_1 + x_2 \geq 10, \\ &\text{Edit } 3'\text{: } 0 \geq 0. \\ &\text{Edit } 4'\text{: } x_3 = 0. \end{aligned} \tag{7}$$

Of course, these edits can be further simplified, Edits $2'$ and $3'$ are obviously redundant. The further simplification for redundant rules will be explained in Section 5.

4. Simplification of Compound Rules

This section deals with the simplification of compound rules by elimination of unnecessary statements. Two new MILP algorithms are presented, based on existing methods from Dillig et al. (2010). The aims are again to improve computational performance and to detect misspecification of edit rules. A possible outcome, especially beneficial to computation performance, is that a conditional rule can be replaced with an unconditional one.

4.1. Implicitly Unsatisfiable Statements

In this subsection compound edit statements of the form (*A* OR *B* OR . . .) are simplified by deletion of statements that cannot be satisfied, given the available set of edit rules. Dillig et al. (2010) call these statements "non-relaxing", since these do not enhance the feasible area of a MILP problem. If, after simplification, only one component remains, a conditional rule has been converted into an unconditional one. An example is

$$\text{Edit 1: } x_1 > 0 \text{ OR } x_2 > 0,$$
$$\text{Edit 2: } x_2 < 0.$$

It is immediately clear that the statement $x_2 > 0$, within Edit 1, cannot possibly be satisfied, because of Edit 2. This statement can be removed from Edit 1, since it is redundant. Consequently, Edit 1 can be formulated as an unconditional rule. A more formal definition is given below:

Definition:
A statement e_{ij} of a compound edit e_i within a feasible edit set $\mathbf{E}$ is implicitly unsatisfiable, if $\mathbf{E} \cup e_{ij}$ is infeasible.

Here, $\mathbf{E} \cup e_{ij}$ 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 $\mathbf{E}$, as if it were a single edit.

An algorithm for removal of implicitly unsatisfiable statements is stated below

Algorithm 1: Identification and removal of implicitly unsatisfiable statements

Input: Feasible edit set $\mathbf{E}$
Output: Feasible edit set $\mathbf{E}$, without implicitly unsatisfiable components.
1 FOR each compound edit $e_i \in \mathbf{E}$ do
2 FOR each statement $e_{ij} \in e_i$ do
3 $\mathbf{E}^* \leftarrow \mathbf{E} \cup e_{ij}$;
4 IF isFeasible($\mathbf{E}^*$) = FALSE THEN $e_i \leftarrow e_i \setminus e_{ij}$
5 NEXT
6 NEXT

In each step one statement of a compound edit is added to a feasible edit set. Subsequently, the feasibility of the extended edit set is checked by isFeasible(), a function that can be implemented by a MILP solver, see Section 2. If the extended edit set is infeasible, the compound edit's statement is implicitly unsatisfiable and therefore redundant.

When applied to our previous example, the algorithm means that the constraints $x_1 > 0$ and $x_2 > 0$ are added to Edits 1 and 2 one by one and that the feasibility is verified for both resulting edit sets. Because the addition of $x_2 > 0$ renders Edits 1 and 2 infeasible, $x_2 > 0$ is an implicitly unsatisfiable statement. It can be deleted from Edit 1 accordingly.

4.2. Implicitly Satisfied Statements

This subsection aims at replacing compound edit rules (*A* or *B* or. . .) with single, unconditional rules. The main idea is that if a compound rule contains a statement (say A)

that is necessarily *True*, the compound rule can be replaced with that single statement. Implicitly satisfied statements are called non-constraining by Dillig et al. (2010), since these do not reduce the feasible region of a MILP problem. Consider the following example:

$$\text{Edit 1: } x_1 < 50 \text{ OR } x_2 > 100,$$
$$\text{Edit 2: } x_1 > 100 \text{ OR } x_2 > 0.$$

For all possible x_1 values, at least one of the statements $x_1 < 50$ and $x_1 > 100$ is not satisfied. Thus, Edits 1 and 2 imply that either $x_2 > 0$, or the even stronger condition $x_2 > 100$, needs to be true. As a result, we obtain that $x_2 > 0$ always needs to hold, in other words $x_2 > 0$ is implicitly satisfied. Consequently, Edit 2 can be replaced with this single statement. A more formal definition is stated below:

Definition:
A component e_{ij} of a compound edit e_i within a feasible edit set $\mathbf{E}$ is implicitly satisfied if $\mathbf{E} \cup \neg\, e_{ij}$ is infeasible (where $\neg$ stands for negation).

This definition makes use of the equivalence of the statements that a compound edit's component is implicitly satisfied and that the opposite of that component cannot occur. An algorithm for identifying implicitly satisfied statements is as follows

Algorithm 2: Identification and replacement of implicitly satisfied statements

Input: Feasible edit set $\mathbf{E}$
Output: Feasible edit set **E,** without implicitly satisfied statements.
1 FOR each compound edit $e_i \in \mathbf{E}$ DO
2 FOR each statement $e_{ij} \in e_i$ DO
3 $\mathbf{E}^* \leftarrow \mathbf{E} \cup \neg\, e_{ij}$;
4 IF isFeasible($\mathbf{E}^*$) = FALSE THEN $\mathbf{E} \leftarrow \{\mathbf{E} \setminus e_i\} \cup e_{ij}$
5 NEXT
6 NEXT

This algorithm has a similar structure as Algorithm 1. Each step of the algorithm checks the feasibility of an extended edit set that is obtained by adding the negation of a statement of a compound rule to the given edits in $\mathbf{E}$. If the resulting edit set turns out to be infeasible, the added statement is "implicitly satisfied". The statement is added to the edit set as an unconditional rule and the conditional rule from which the statement is obtained is deleted.

When applied to our previous example, the constraints $x_1 \geq 50, x_2 \leq 100, x_1 \leq 100$ and $x_2 \leq 0$ are added to Edits 1 and 2 one by one, which are the negations of the original edit components. Feasibility is checked for all resulting edit sets. Because the addition of $x_2 \leq 0$ renders Edits 1 and 2 infeasible, $x_2 > 0$ is implicitly satisfied. Hence, Edit 2 can be replaced with the unconditional rule $x_2 > 0$.

5. Redundant Edit Removal

This section's aim is to simplify edit sets by removal of redundant edits, that is, rules that can be left out of an edit set, without affecting the set of feasible records. The removal of

redundant constraints may speed up the error correction process without losing power of correction. Because redundant edits may emerge as a result of fixed value substitution and simplification of conditional edits, it is important that redundant edit removal is conducted after these other steps. Consider the following example:

$$\begin{aligned}
&\text{Edit 1: } x_1 + x_2 \leq T_1,\\
&\text{Edit 2: } x_3 + x_4 \leq T_2,\\
&\text{Edit 3: } T_1 + T_2 = T_3,\\
&\text{Edit 4: } x_1 + x_2 + x_3 + x_4 \leq T_3.
\end{aligned}$$

Edit 4 can be omitted because it is implied by Edits 1, 2, and 3.

An edit is redundant if other edits imply that the edit is 'always satisfied'. As mentioned in Subsection 4.2, this is equivalent to the statement that the negation of the edit cannot occur. This leads to the following definition

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

The edit set $\{\boldsymbol{E} \backslash e_i\} \cup \neg\, e_i$ is obtained from **E**, by replacing Edit e_i by its negation.

In literature many methods have been mentioned for detection of redundant constraints. Paulraj and Sumathi (2010) performed a comparative study. Below we describe a method mentioned by for example Felfernig et al. (2011), Chmeiss et al. (2008) and Bruni (2005). The reason for choosing this method is its simplicity and the possibility of implementing it by a MILP solver.

Algorithm 3: Identification and removal of redundant edits

Input: Feasible edit set **E**
Output: Feasible edit set **E**, without redundant edits
1 FOR each Edit $e_i \in \mathbf{E}$ DO
2 $\quad \mathbf{E}^* \leftarrow \{\mathbf{E} \backslash e_i\} \cup \neg\, e_i$;
3 $\quad$ IF isFeasible($\mathbf{E}^*$) = FALSE THEN $\mathbf{E} \leftarrow \mathbf{E} \backslash e_i$
4 NEXT

When applied to previous example, the algorithm means that the negations of Edits 1, 2, 3, and 4 are added to the edit set one by one and that the feasibility is verified for all of the resulting set of rules. In this way, the redundancy of Edit 4 can be easily demonstrated.

Below a few words on the computation of negations. The negation of an equality constraints can be expressed as combination of two inequality constraints. For example, in previous example the negation of Edit 3, can be expressed as $T_1 + T_2 < T_3$ OR $T_1 + T_2 > T_3$. These two constraints are added to the three original rules one by one. Only if both additions lead to infeasible edit sets, one could conclude that Edit 3 is redundant. In our example, Edit 3 is however clearly not redundant.

The negation of a compound edit rule e_i, expressed as the disjunction $\cup_{j=1}^{D_i}\left(\left(\boldsymbol{a}_{ij}^{C}\right)^{T}\mathbf{x} \leq b_{ij}^{C}\right)$, is given by,

$$\bigcap_{j=1}^{D_i}\left(\left(\boldsymbol{a}_{ij}^{C}\right)^{T}\mathbf{x} > b_{ij}^{C}\right) \quad j = 1, \ldots, D_i,$$

a combination of D_i linear, unconditional constraints that all have to be satisfied.

6. Applications

Aim of this section is to apply constraints simplification methods on 'real-life' edit sets. We would like to show that these edit sets can actually be simplified. Moreover, we demonstrate that constraint simplification improves data editing's performance.

All applications were performed on a 32-bit Windows 7 laptop with a 2.80 GHz CPU and 3 Gigabyte of RAM memory. The methods from Sections 3–5, were implemented by R. The free LpSolveAPI was used as a MILP solver (Konis 2016) and the Editrules package (De Jonge and Van der Loo 2015) was implemented for automatic data editing. The following edit sets were considered:

1. Sales: Real-life edit set used for the 2012 Dutch Structural Business Statistics for sale of motor vehicles for businesses with fewer than ten employed persons;
2. Maintenance: Real-life edit set used for the 2012 Dutch Structural Business Statistics for maintenance of motor vehicles for businesses with fewer than ten employed persons;
3. Health-care: Edit set under development, meant to be used for a Dutch survey among welfare and childcare institutions;

All methods for constraint simplification in Sections 3–5 were applied to these three data sets. Automatic data editing was applied to the first two edit sets only, because of the lack of data for the third application.

Table 1. Results of three real-life applications.

	Sales	Maintenance	Health-care
Original edits			
Number of edits	115	119	196
-of which conditional:	26	29	114
Number of variables in edits	74	74	75
Simplification			
Fixed values	3	7	2
Conditional edits			
-Implicitly unsatisfiable components	1	1	4
-Implicitly satisfied components	1	1	3
Redundant edits	22	29	10
-of which conditional	7	13	3
Cleaned edits			
Number of edits	93	90	186
-of which conditional:	19	16	104
Computation Time (in seconds)	5	6	2,465

Table 2. Automatic data editing, original and simplified edit sets.

	Sales ($N = 614$ records)		Maintenance ($N = 197$ records)	
	Original edits	Simplified edits	Original edits	Simplified edits
Processed records*	613	613	197	197
Total time (in seconds)	2,639	2,039	479	217
Number records $<$ 10 sec	592	598	191	194

* = given a maximum computation time of ten seconds per record.

Firstly, Table 1 shows the feasibility of constraint simplification on a regular computer. One could note that computation time for the third application is relatively large, about 40 minutes, which can be explained from the many conditional rules. Large computation time is however not a problem, because edit rules simplification only needs to be conducted once, after designing or revising an edit set.

Secondly, Table 1 demonstrates that all simplification features in Sections 3–5 are useful, as each feature actually simplifies all three edit sets. The total number of edit rules is reduced by 5–20%; the number of conditional edits by 10–45%.

Table 2 shows that total computation time for automatic data editing is reduced by 23% for the Sales application and even by 55% for the Maintenance data set. The latter reduction can be largely attributed to only one record, whose computation times are 297 and 109 seconds for the original and simplified edit sets. This actually points out that the worst-case performance is important in data editing, but also shows that worst-case performance can be noticeably improved by rule simplification.

A practical solution to the possibly long computation time is to limit the available time for each record. The last row in Table 2 shows that edit rule simplification slightly increases the amount of records that are processed within ten seconds.

7. Discussion

Many works from the literature present automatic constraint simplification techniques that are able to greatly improve computational performances of large optimization problems. But, to the best knowledge of the author, these techniques are not often applied in the field of data editing.

This article shows that automated data editing can benefit from constraint simplification. A number of methods was presented for numerical data, based on MILP programming. Much attention was given to conditional IF-THEN rules that often occur in official statistics and that are particularly important for computational performance.

The feasibility of constraint simplification was demonstrated on a regular computer using freely available MILP solvers. It was shown that real-life edit sets can actually be simplified. As a result, the total computation time for localising erroneous values was reduced up to 55%; a reduction that can be mainly attributed to a few records with the

largest computation time. Hence, constraint simplification is an important step in further enhancing the practicality of automatic data editing.

Another benefit is that constraint simplification provides insight in the joint consequences of a set of rules. Manual inspection of automatically determined redundant rules and variables with a fixed value or finite bounds might reveal errors in rule formulation. Correction of these errors increases the quality of automated data editing and reduces the need for manual correction of automatically edited data.

A practical merit of the proposed methods 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.

This article implicitly assumed that edits are interconnected. However, if this is not the case, it is advisable to split an edit set $\mathbf{E}$ into disjunct sets, $\oplus_i \mathbf{E}_i$, such that $e_i \in \mathbf{E}_i$ and $e_j \in \mathbf{E}_j$ $(i \neq j)$ do not have any variable in common. Disjunct edit sets can be treated independently, which may improve performance of both data editing and edit rule simplification.

The simplification methods in this article have been designed for feasible edit sets. Despite that infeasible edit rules are useless for practical application, infeasible rules may occur in practise, for instance due to misspecification. In general, it can be hard to find the cause of a contradiction, especially if the number of edit rules is large. Therefore, most methods for dealing with inconsistency concentrate on isolating a smallest possible subset of inconsistent edit rules: a so-called irreducible inconsistent subset (IIS). Several algorithms for detecting IIS's are available from literature. The so-called "Deletion Filter" by Chinneck (1997) can be advised for many applications as it is easily understood, suitable for conditional "IF-THEN" edits and applicable for MILP programming. In a recent publication, Bruni and Bianchi (2012) proposed another, innovative approach, based on Farka's lemma. Their method however relies on an assumption, the so-called Integral Point property, that is unknown to be true for general applications.

A direction for further research is to introduce more constraint simplification techniques for data editing. In this article we considered numerical data. Methods for categorical data could be developed in the future.

8. References

Banff Support Team. 2008. *Functional Description of the BANFF System for Edit and Imputation*. Ottawa: Statistics Canada (Technical report).

Bertsimas, D. and J.N. Tsitsiklis. 1997. *Introduction to Linear Optimization*. Nashua: Athena Scientific.

Bruni, R. 2005. "Error Correction for Massive Data Sets." *Optimization Methods and Software* 20: 291–310. Doi: http://dx.doi.org/10.1080/10556780512331318281.

Bruni, R. and G. Bianchi. 2012. "A Formal Procedure for Finding Contradictions into a Set of Rules." *Applied Mathematical Sciences* 6: 6253–6271.

Chen, D., R.G. Batson, and Y. Dang. 2010. *Applied Integer Programming; Modelling and Solution*. Hoboken: John Wiley & Sons. Doi: http://dx.doi.org/10.1002/9781118166000.

Chinneck, J.W. 1997. "Finding a Useful Subset of Constraints for Analysis in an Infeasible Linear Program." *INFORMS Journal on Computing* 9: 164–174. Doi: http://dx.doi.org/10.1287/ijoc.9.2.164. Available at: http://www.sce.carleton.ca/faculty/chinneck/docs/UsefulSubset.pdf (accessed January 2017).

Chmeiss, A., V. Krawczyk, and L. Sais. 2008. "Redundancy in CSPs." In Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008), August 21–25, 2008. Patras, Greece. Amsterdam: IOS Press. Doi: http://dx.doi.org/10.3233/978-1-58603-891-5-907.

De Jonge, E. and M. van der Loo. 2015. *Editrules: R Package for Parsing and Manipulating of Edit Rules and Error Localization*. R Package Version 2.9-0. Available at: http://cran.r-project.org/package=editrules (accessed May 2017).

De Waal, T., J. Pannekoek, and S. Scholtus. 2011. *Handbook of Statistical Data Editing and Imputation*. New York: John Wiley & Sons. Doi: http://dx.doi.org/10.1002/9780470904848.

Dillig, I., T. Dillig, and A. Aiken. 2010. "Small Formulas for Large Programs: On-Line Constraint Simplification in Scalable Static Analysis." In Proceedings of the 17th international conference on Static analysis (SAS'10), September 14–16, 2010 Perpignan, France. Berlin Heidelberg: Springer-Verlag. Available at: http://theory.stanford.edu/~aiken/publications/papers/sas10.pdf (accessed January 2017).

Felfernig, A., C. Zehentner, and P. Blazek. 2011. "CoreDiag: Eliminating Redundancy in Constraint Sets." Proceedings of 22nd International Workshop on Principles of Diagnosis, October 4–7, 2011, Murnau, Germany. Available at: http://www.ist.tugraz.at/felfernig/images/stories/home/dx_corediag.pdf (accessed March 2017).

Fellegi, I.P. and D. Holt. 1976. "A Systematic Approach to Automatic Edit and Imputation." *Journal of the American Statistical Association* 71: 17–35. Doi: http://dx.doi.org/10.1080/01621459.1976.10481472.

Hooker, J. 2000. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. New York: John Wiley & Sons. Doi: http://dx.doi.org/10.1002/9781118033036.

Konis, K. 2016. *lpSolveAPI: R Interface for lpsolve. Version 5.5.2.0-17* R package version 5.5.2.0. Available at: https://cran.r-project.org/web/packages/lpSolveAPI/index.html (accessed January 2017).

Pannekoek, J., S. Scholtus, and M. van der Loo. 2013. "Automated and Manual Data Editing: a View on Process Design and Methodology." *Journal of Official Statistics* 29: 511–537. Doi: http://dx.doi.org/10.2478/jos-2013-0038.

Paulraj, S. and P. Sumathi. 2010. "A Comparative Study of Redundant Constraints Identification Methods in Linear Programming Problems." *Mathematical Problems in Engineering*. Article ID 723402. Doi: http://dx.doi.org/10.1155/2010/723402.

Piette, C. 2008. "Let the Solver Deal with Redundancy." In Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08), November 3–5, 2008, Dayton, Ohio. Washington DC: IEEE Computer Society. Doi: http://dx.doi.org/10.1109/ICTAI.2008.38. Available at: https://hal.archives-ouvertes.fr/hal-00865304/document (accessed March 2017).

Telgen, J. 1983. "Identifying Redundant Constraints and Implicit Equalities in Systems of Linear Constraints." *Management Science* 29: 1209–1222. Doi: http://dx.doi.org/10.1287/mnsc.29.10.1209.

Received March 2016
Revised August 2017
Accepted September 2017