Efficient edit rule implication for nominal and ordinal data

Toon Boeckling[a,*], Guy De Tré[a], Antoon Bronselaer[a]

[a] *Ghent University, Department of Telecommunications and Information Processing, St.-Pietersnieuwstraat 41, B-9000, Ghent, Belgium*

Abstract

Edit rule implication is an essential subtask when repairing data inconsistencies against a set of edit rules. In this paper, novel techniques to enhance the performance of this subtask are studied. Our work includes several contributions. First, we draw attention to the case of nominal edit rules in particular. We point out that in many cases, starting with a set of edit rules that is as small as possible is important to improve the performance. This could be achieved by folding edit rules together. Besides that, an enhanced nominal edit rule implication algorithm is proposed, exploiting the properties of nominal edit rules. Second, we introduce ordinal edit rules as a generalization of nominal edit rules, used to capture data inconsistencies for data measured on an ordinal scale and we propose an ordinal edit rule implication algorithm. Evaluation of our methods shows promising results for both implication algorithms, with the ordinal algorithm as best performing in general. On average, our techniques improve the state-of-the-art algorithm for edit rule implication with more than 50%.

Keywords: Data Quality, Edit Rules, Rule Implication

1. Introduction

The last decades, data quality is a topic that is gaining importance as it is an essential part in the process of data management and is especially rel-

*Corresponding author

Email addresses: `toon.boeckling@ugent.be` (Toon Boeckling), `guy.detre@ugent.be` (Guy De Tré), `antoon.bronselaer@ugent.be` (Antoon Bronselaer)

evant in the case of large, heterogeneous data volumes. Although, there are many facets related to this topic, such as completeness, accuracy and currency, one of the main problems is to safeguard *consistency* of data [3, 4, 9, 21, 42]. In this paper, we will focus on a rule-based approach for consistency maintenance. Many rule-based mechanisms have been proposed in the past, such as functional dependencies [1, 6], conditional functional dependencies [7, 22], inclusion dependencies [6], denial constraints [15] and pattern functional dependencies [40]. Although these types of constraints tend to capture most of the consistency problems due to their extensive expressiveness, studying and using them in a practical fashion can become a task of high complexity, especially when it comes to repairing violations. A first algorithm that is studied to resolve this task is the Chase algorithm [27], but a particular problem is that its search space rapidly tends to increase when used in combination with high-expressive constraints and one has to rely on heuristics. A second tool is HoloClean [43], which features probabilistic models to find repairs and training such a model can also become intensive. With these issues in mind, the focus here is on more simple types of constraints, denoted as *tuple-level constraints*. Although they are less expressive than the earlier mentioned types, a recent study argues that a large portion of inconsistencies in real-life data sets can be captured by them [41].

A leading contribution to tuple-level constraints has been made by Fellegi and Holt, who introduce the concept of *edit rules* [23]. Informally, edit rules (edits in short) model in a concise way all data objects which are not permitted to exist in a dataset (e.g. according to the Belgian Constitution, people under 18 years old cannot be married). Within the framework of Fellegi and Holt, the properties of edit rules can be exploited easily to repair violations while minimizing a linear repair cost function [8, 19, 23, 38, 39]. In other words, one can easily identify a minimal set of attributes in an inconsistent data object for which the values have to be changed in order to make it consistent. This statement, however, only stands if sufficient information is captured by the given set of edit rules to which the object is inconsistent and no additional implicit

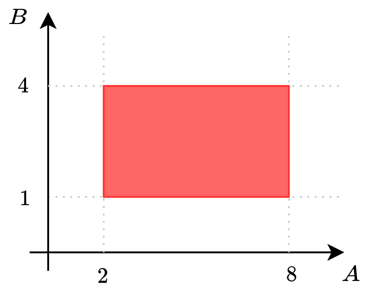


Figure 1: A simple example constraint that cannot be written as a linear edit rule. The red area models non-permitted values.

information can be derived. Indeed, often it is possible that combinations of edit rules account for implication of new edit rules. These edit rules are indispensable because they reveal extra information that is useful for finding a minimal set of attributes to repair. Although there are many papers that study implication algorithms [8, 19, 25, 26, 32, 33, 34, 46, 47, 48], it still remains an exponential problem as it relies on combinations of edit rules. Because of this reason, edit rule implication can be enhanced further to make it more practically useful, especially for large problems.

Furthermore, edit rules are initially introduced for data measured on a nominal scale, in which case they are usually represented as a cross-product of subsets of the attribute domains [23]. For data that are measured beyond the nominal scale, edit rules in the form of linear (in)equalities have been studied only [19, 26]. Many concepts introduced for nominal edits can then be transferred to linear edit rules quite straightforwardly. Edit rule implication is less obvious to transfer, but comes down to variable substitution in the case of equality and Fourier-Motzkin elimination in the case of inequality [19]. The concept of linear edit rules however comes with some issues. A first problem is that implication becomes complicated when a mixture of nominal and linear edit rules is used [19]. In practice, not all necessary edit rules are implied and one resorts to branch-and-bound methods that rely on 'local' edit rule implication for each erroneous data object independently. These methods can however have scaling issues when the number of edits or errors within one data object gets large. A second problem (actually, a problem by design) is that linear edit rules rely on the prerequisite that (non)-permitted combinations of values are expressed by means of a linear connection. This hinges on two implicit assumptions, the first being that data are additive and the second being that interactions between variables are linear. We argue that there are simple, yet non-linear connections that are impossible to model by this constraint. For example, the constraint on two domains D_1 and D_2 as shown in Figure 1 cannot be written as a linear edit.

In order to further improve edit rule implication and help to solve the above mentioned problems, the following contributions are provided.

1. The properties of nominal edit rules are exploited further to enhance the edit rule implication algorithm with more efficient pruning strategies.
2. Edit rules for ordered data are investigated and a novel ordinal edit rule implication algorithm is proposed and shown to be more efficient than the existing algorithm for nominal data.

The remainder of the paper is structured as follows. In Section 2, we recall the concepts related to the Fellegi-Holt framework. After this, in Section 3, we study near-optimal representations of tuple-level constraints by sets of edit rules to enhance the performance of edit rule implication. An improved implication algorithm, relying on the properties of nominal edit rules, is proposed in Section 4. Then, in Section 5, ordinal edit rules are defined and an ordinal implication algorithm is proposed. In Section 6, an overview is given of the state-of-the-art literature related to the topics studied in this paper. We set up several experiments to evaluate and validate the practicality of our contributions in Section 7 and finally, in Section 8, we conclude our work and give a brief overview of future research directions.

2. The Fellegi-Holt framework

Before the main goal of this paper is addressed, some theoretical and practical preliminaries concerning the Fellegi-Holt framework [23] are given.

Definition 1 (Nominal edit rule)**.** *A* nominal edit rule E^k *(where k is an index to identify the edit) over a set of m attributes*[1] $\mathcal{R} = \{1, \ldots, m\}$*, with respective finite domains $D_1, \ldots, D_m$, covers a subset $P(E^k)$ of the universe $D_1 \times \ldots \times D_m$, which is defined by*

$$P(E^k) = \prod_{j=1}^{m} A_j^k \tag{1}$$

with $A_j^k \subseteq D_j$ for each attribute $j \in \{1, \ldots, m\}$.

Semantically, an edit rule defines a subset of the universe such that elements of this subset are forbidden to exist in a consistent data set. Therefore, notice that edit rules belong to the category of tuple-level constraints. Formally, this means that, if a data object $o = (v_1, \ldots, v_m) \in P(E^k)$, with $v_j \in D_j$ for each $j \in \{1, \ldots, m\}$, then edit E^k is *failed* by o and o is inconsistent according to this edit. This is denoted by $o \not\models E^k$. Otherwise, o *satisfies* rule E^k, which is denoted by $o \models E^k$. It is said that an attribute j *enters* or *is involved in* an edit rule E^k if $A_j^k \subset D_j$, which implies that violation of E^k may depend on j. The set of attributes entering in E^k is denoted by $\mathcal{I}(E^k)$. If an edit rule involves a single attribute, it represents a constraint on the domain of that attribute: some values in the domain are then a priori not permitted. If there exists an attribute j in E^k for which $A_j^k = \emptyset$, then E^k is a *tautology* because it is always satisfied.

In Table 1, an example of six edit rules over four attributes, based on clinical trial design data[2], is given. The domains of the attributes are: $D_{\texttt{arms}} = \{1, 2, 3\}$, $D_{\texttt{parallel}} = \{\text{No}, \text{Yes}\}$, $D_{\texttt{crossover}} = \{\text{No}, \text{Yes}\}$ and $D_{\texttt{model}} = \{\text{S}, \text{P}, \text{X}\}$. For E^1, we see that the involved attributes are given by the set $\mathcal{I}(E^1) = \{\texttt{arms}, \texttt{model}\}$ and the edit states that value 1 of attribute `arms` is not allowed to appear together with values P or X of attribute `model`, regardless of the values of the other two attributes.

[1]For the sake of simplicity of notation, each attribute is represented by a unique index.

[2]Attribute semantics: `arms`: number of participant groups, `parallel`: all arms receive one separate treatment, `crossover`: all arms receive all treatments, `model`: S(ingle) treatment, P(arallel), X (crossover).

Table 1: A four attribute, six edit example based on clinical trial design data.

	`arms`	`parallel`	`crossover`	`model`
E^1	$\{1\}$	$D_{\texttt{parallel}}$	$D_{\texttt{crossover}}$	$\{\text{P, X}\}$
E^2	$D_{\texttt{arms}}$	$\{\text{Yes}\}$	$D_{\texttt{crossover}}$	$\{\text{S, X}\}$
E^3	$D_{\texttt{arms}}$	$\{\text{No}\}$	$D_{\texttt{crossover}}$	$\{\text{P}\}$
E^4	$D_{\texttt{arms}}$	$D_{\texttt{parallel}}$	$\{\text{No}\}$	$\{\text{X}\}$
E^5	$D_{\texttt{arms}}$	$D_{\texttt{parallel}}$	$\{\text{Yes}\}$	$\{\text{S, P}\}$
E^6	$\{2, 3\}$	$D_{\texttt{parallel}}$	$D_{\texttt{crossover}}$	$\{\text{S}\}$

A fundamental problem is to identify, for each failed data object o in a data set, a *solution* $\mathcal{S} \subseteq \mathcal{R}$ such that, after adapting the values of $\mathcal{S}$ in o, the new object o' does not fail any edits. A solution is called *minimal* if it minimizes a linear repair cost function. Although there are many possible cost functions, we restrict ourselves to constant costs of 1 to repair each attribute. The resulting data object o' is called a *repair*. For example, data object $o =$ (2, Yes, Yes, P) fails edit E^5 given in Table 1. When assuming the abovementioned cost function, a minimal solution to repair o is $\mathcal{S} = \{\texttt{crossover}\}$ because a possible repair is $o' = (2, \text{Yes}, \underline{\text{No}}, \text{P})$.

In order to easily find solutions, Fellegi and Holt introduced the concept of *implied* edit rules. For these edits, they proved that, when added to the set of *explicit* (or given) edits, all minimal set covers of failing edits (where a set covers a failing edit if at least one involved attribute of that edit is in the set) are also minimal solutions [23]. In other words, implied edit rules do not capture new inconsistencies, but they are important to generate when one is interested in correctly localizing the attributes $\mathcal{S}$ that are in error in a data object o by means of the set cover method. In the remainder we will call sets of edit rules meeting this requirement, *sufficient sets*. Given a set of edits $\mathcal{E}$, it is possible to generate implied edit rules from this set by using the following lemma.

Lemma 1 (Implied edits). *For a given* contributing set *of edit rules $\mathcal{E}_c \subseteq \mathcal{E}$ over $\mathcal{R} = \{1, \ldots, m\}$ and a* generating attribute *(or* generator*) $g \in \mathcal{R}$, the edit rule E^* for which*

$$A^*_j = \bigcap_{E^k \in \mathcal{E}_c} A^k_j, \qquad \forall j \in \mathcal{R}, j \neq g$$

$$A^*_g = \bigcup_{E^k \in \mathcal{E}_c} A^k_g$$

is called an implied edit if E^* is not a tautology.

Note that a contributing set consisting of only one edit rule leads to the degenerate case where this edit implies itself. Therefore, it is assumed in the remainder that each contributing set consists of at least two edit rules. Besides that, we will refer to the construction of an implied edit rule by means of a contributing set $\mathcal{E}_c$ and generator g with $\mathrm{FH}(g, \mathcal{E}_c)$ in short.

One way of generating all implied edit rules is by repeatedly applying Lemma 1. However, because this task is of exponential complexity, more efficient ways to generate sufficient sets are needed. The best known state-of-the-art algorithm for generating a sufficient set of edit rules is the Field Code Forest (FCF) algorithm[3] [8, 25]. Two steps are repeated during the course of this algorithm, which are (1) the selection of a generator g dictated by an FCF data structure formed by all attribute combinations and (2) the generation of implied edit rules by means of a generator g, called edit (rule) implication in the remainder. To reduce the number of implied edit rules, the algorithm should only keep track of necessary edit rules (and not necessarily *all* implied edits). These necessary edit rules have two important properties, as they are: (1) *(essentially) new* and (2) *non-redundant* (NNR edit rules in short) and the fact that these conditions are necessary and sufficient is proven extensively in [8, 19, 23, 25].

Essentially new edit rules are defined as follows.

Definition 2 (Essentially new edits). *An implied edit rule E^* is called* essentially new *iff the generator g enters in each of the edit rules in the contributing*

[3]The correctness of this algorithm is extensively proven in [8]

set (i.e. $g \in \mathcal{I}(E^k)$, $\forall E^k \in \mathcal{E}_c$), but does not enter in E^* (i.e. $g \notin \mathcal{I}(E^*)$).

Besides that, *redundant* edit rules are defined as follows.

Definition 3 (Redundant edits)**.** *An edit rule E^r is* redundant *to an edit rule E^d iff $P(E^r) \subseteq P(E^d)$. It is said that E^d dominates E^r.*

An example of an essentially new edit rule is $E^* = \{1\} \times \{\text{Yes}\} \times D_{\texttt{crossover}} \times D_{\texttt{model}}$, which can be implied by using $\{E^1, E^2\}$ as contributing set and attribute `model` as generator. The edit rule $\{1\} \times \{\text{Yes}\} \times \{\text{No}\} \times D_{\texttt{model}}$ is redundant to E^*.

In this paper, an important issue with the FCF algorithm is addressed, which is the lack of efficient strategy for edit rule implication given a generator g and contributing set $\mathcal{E}_c$. A first optimization is to reduce the number of implied edit rules and to generate only NNR edit rules [19, 25]. A consequence of this reduction and Definition 2 is that only combinations of edits in which attribute g enters should be tested to see if they lead to an essentially new edit rule. In the remainder of this paper, we will denote these edit rules as *candidate contributors for generator g*. One can verify that, if there are n_g candidate contributors for generator g, an upper bound on the number of combinations to test is

$$\sum_{i=2}^{n_g} \binom{n_g}{i} = \sum_{i=0}^{n_g} \binom{n_g}{i} - \binom{n_g}{1} - \binom{n_g}{0} = 2^{n_g} - n_g - 1 \qquad (2)$$

as no combinations of 0- and 1-sets will lead to an NNR edit rule. Because this number still grows exponentially in terms of n_g, it is important to reduce this number further in any possible way.

3. Explicit set representations

The representation of tuple-level constraints by means of an explicit set of edit rules is not necessarily unique. Therefore, we study the issues related to this observation in 3.1 and we provide techniques in 3.2 and 3.3 to reduce the number of edit rules in an explicit set in a near-optimal way, in order to enhance edit rule implication.

3.1. Problem statement

A logical first step when addressing the issues of the FCF algorithm is to investigate the explicit set of edit rules that is passed as input parameter. This explicit set defines all value combinations that are forbidden to appear within a consistent data set. A problem is that the representation of these forbidden value combinations by means of an explicit set of edits is not necessarily unique. As an example, it is straightforward to see that any set of forbidden value combinations can be represented by a set, in which each edit rule covers only one element of the forbidden subset. This implies that, given such an explicit set, each attribute will enter in each edit rule (with a singleton value set) and, when one seeks to construct all NNR edits for a given generator in a naive way, all combinations of all edits should be tested for contributing to the generation of an NNR edit rule. According to Eq. 2, this could increase the number of combinations to test enormously. Considering the example given in Table 1, the original explicit set covers 31 unique forbidden elements in the universe, which could possibly result in 31 single-element edit rules, all consisting of four singletons (one for each attribute). This is an increase of 25 edit rules and 112 entering attributes in total in comparison to the given explicit set.

This simple example shows that considering the most optimal representation of forbidden value combinations will improve the performance of the FCF algorithm undoubtly. According to Eq. 2, the reference *most optimal* indicates a representation with as few edits and as few entering attributes as possible. One might think that defining an optimal explicit set is the responsibility of the subject matter expert, but often it is the case that such an expert is not involved (e.g. when edits are automatically discovered [5, 41]). Besides that, constructing such an optimal representation is an NP-hard optimization problem. This is because solving this problem can be tackled as a variation of solving the NP-hard problem of deciding the *edge clique cover number* in graph theory [24, 28]. To overcome this problem, a rather simple, heuristic approach, which aims to find a representation close to the optimal one, is proposed in 3.2 and 3.3.

3.2. Edit rule folding

As stated in 3.1, decreasing the total number of edits and entering attributes will positively impact the performance of edit rule implication and as a result, of the FCF algorithm. A possible way to achieve this is by folding edits in a given explicit set together. Before investigating the properties of edit folding, the construction procedure of a folded edit rule E^f from a set of edits $\mathcal{E}_f$ is given by the following lemma.

Lemma 2 (Folded edits)**.** *For a given set of edits $\mathcal{E}_f$ over $\mathcal{R} = \{1, \ldots, m\}$ and an attribute $c \in \mathcal{R}$, the edit rule E^f for which*

$$\begin{aligned} A_j^f &= A_j^k \qquad \forall j \in \mathcal{R}, j \neq c \wedge \forall E^k \in \mathcal{E}_f \\ A_c^f &= \bigcup_{E^k \in \mathcal{E}_f} A_c^k \end{aligned}$$

is called a folded edit rule*.*

Lemma 2 states that a set of edit rules over m attributes can be folded if and only if these edits have exactly the same value sets for at least $m-1$ attributes. $\mathcal{E}_f$ is called the *folding set* and c is called the *folding attribute* needed to construct E^f. For example, edits $E^{61} = \{2, 3\} \times \{\text{No}\} \times D_{\texttt{crossover}} \times \{\text{S}\}$ and $E^{62} = \{2, 3\} \times \{\text{Yes}\} \times D_{\texttt{crossover}} \times \{\text{S}\}$ can be used as folding set to construct edit E^6 defined in Table 1 by using folding attribute `parallel`.

To investigate the profit of substituting a folding set with the folded edit, suppose a folding set $\mathcal{E}_f$ of edit rules over m attributes is given with attributes $\{j_1, \ldots, j_i\}, i \leq m$ entering in each edit. According to Lemma 2, the folded edit consists of at most i (resp. at least $i-1$) entering attributes, which implies a reduction of $|\mathcal{E}_f| - 1$ edits and $i * (|\mathcal{E}_f| - 1)$ (resp. $i * (|\mathcal{E}_f| - 1) - 1$) entering attributes. Besides that, using folded edits instead of their folding sets as input to the FCF algorithm will still result in a sufficient set. The reason for this is that a folded edit dominates all edits in the folding set from which it is constructed and redundant edits can be discarded during generation [8, 25]. Based on the aforementioned findings, a heuristic algorithm that creates a near-optimal folded set of edits from a given explicit set is proposed in 3.3.

3.3. A heuristic explicit set folding algorithm

In the following, a non-overlapping, heuristic set folding algorithm is described, of which the pseudocode is given in Algorithm 1. By *non-overlapping*, it is meant that, when folding the edits, no additional duplicate elements in the entire forbidden subset are created other than the ones that initially exist. Given the edits in Table 1, an example of a duplicate element is (1, No, No, P), because it is covered by both E^1 and E^3. The reason to consider a non-overlapping set folding algorithm is to keep the procedure as simple as possible.

Algorithm 1 A heuristic explicit set folding algorithm.

```
1: function foldExplicitSet($\mathcal{E}$)
2:     totalEntAttr ← $\sum_{E^k \in \mathcal{E}} |\mathcal{I}(E^k)|$
3:     maxDiff ← 0, bestSet ← null
4:     for $j \in \{1, \ldots, m\}$ do
5:         $\mathcal{E}'$ ← foldEdits($\mathcal{E}$, $j$)
6:         newTotalEntAttr ← $\sum_{E^k \in \mathcal{E}'} |\mathcal{I}(E^k)|$
7:         newDiff ← totalEntAttr − newTotalEntAttr
8:         if newDiff > maxDiff then
9:             maxDiff ← newDiff
10:            bestSet ← $\mathcal{E}'$
11:    if maxDiff = 0 then
12:        return $\mathcal{E}$
13:    else
14:        return foldExplicitSet(bestSet)
```

The algorithm takes as input parameter a predefined explicit set of edit rules $\mathcal{E}$ over m attributes $\mathcal{R} = \{1, \ldots, m\}$. It searches in each (recursive) step for a local optimum as it tries to identify the folding attribute that maximizes the profit in terms of total number of entering attributes within the edits. This is done by applying the following procedure recursively. First, the total number of entering attributes over all edits in $\mathcal{E}$ is counted (line 2) and the optimization variable **maxDiff** is initialized (line 3). Then, each attribute j is tested as folding attribute and, starting from the current set of edits $\mathcal{E}$, as many folded edits as possible are constructed by using this attribute (line 5). This is done by the **foldEdits** procedure which applies the construction procedure proposed in Lemma 2 for each possible folding set resulting in a folded edit. The edits in $\mathcal{E}$ used in the folding sets are replaced by their folded edit and the resulting set is denoted by $\mathcal{E}'$. The profit of using attribute j as folding attribute is calculated as the difference between the previous number of entering attributes (in $\mathcal{E}$) and the new number of entering attributes (in $\mathcal{E}'$) (line 6-7). If the profit exceeds the current most optimal profit, $\mathcal{E}'$ is kept as new **bestSet** (line 8-10). Because it is possible to fold multiple times on the same folding attribute, the only condition that leads to completing the algorithm is when no additional profit can be reached (**maxDiff** $= 0$, line 11) when using any folding attribute.

To end this section, a complexity analysis of the algorithm is presented. Suppose therefore that an explicit set of n edits over m attributes is given as input parameter to the algorithm. In a worst case scenario, the **foldExplicitSet** procedure is called n times and in each call, only two edits are folded, such that the new best set of edits contains one edit less than the previous set. During each call, m attributes are tested as folding attribute, which are used during the **foldEdits** procedure, for which an optimal implementation has linear time complexity in function of n. Therefore, the worst case theoretical complexity of the entire algorithm is $O(mn^2)$. In practice, we can expect the performance to be much better as often, the **foldEdits** procedure executed with the best folding attribute (in terms of profit) potentially uses more than one folding set, each containing more than two edits. Besides that, it is not uncommon that there are edits in the given explicit set that cannot be combined with any other edit. Finally, the number of edits n in the explicit set decreases with each call, particularly because of the edit folding. These reasons potentially ensure that the stop condition will be reached sooner than only after n calls. A thorough analysis of the practical performance of this algorithm is given in Section 7.

4. Nominal edit rule implication

In this section, we study the efficient retrieval of (potential) NNR edit rules by means of a given generator. Indeed, as stated in Section 2, it is unnecessary that combinations of edit rules that will never lead to an NNR edit rule are tested, but the question remains how one can efficiently detect those combinations. Therefore, three rules that can be used to prune combinations of edit rules that certainly will not lead to the generation of an NNR edit, are proposed in 4.1. In 4.2, an efficient nominal edit rule implication algorithm, which exploits the proposed pruning rules, is proposed.

4.1. Pruning rules

A first optimization to reduce the number of combinations to test during edit rule implication is to test only combinations of candidate contributors for generator g (Section 2). Besides that, the following three additional properties of NNR edit rules and their contributing sets can be used to prune combinations during edit rule implication.

Proposition 1 (Pruning rule 1)**.** *Any superset $\mathcal{E}_{c'}$ of a contributing set $\mathcal{E}_c$ that leads to the generation of a tautology $E^* = \emptyset$ by means of a generator g, will also lead to the generation of a tautology E^{**} by means of g.*

Proof. Because $\mathcal{E}_c$ leads to the generation of a tautology $E^* = \emptyset$ by means of generator g, the value set of at least one of the attributes $j \in \mathcal{R} \setminus g$ is empty ($A^*_j = \emptyset$). According to Lemma 1, if one adds any additional edit rule E^k to $\mathcal{E}_c$ in order to generate edit rule E^{**} by means of g, the resulting value set A^{**}_j will remain empty, because $A^*_j \cap A^k_j = \emptyset$. □

Proposition 1 states that, if a combination is found leading to a tautology, none of its supersets will ever lead to an NNR edit rule, reducing the total number of combinations to test.

Proposition 2 (Pruning rule 2)**.** *Any superset $\mathcal{E}_{c'}$ of a contributing set $\mathcal{E}_c$ that leads to the generation of an essentially new edit rule E^* by means of a generator*

g, will also lead to the generation of E^* or an edit rule E^{**} that is redundant to E^* by means of g.

Proof. This proposition is stated in [23], pg. 29 and in [25], pg. 746 $\square$

Proposition 2 states that, if a combination is found leading to an essentially new edit rule, none of its supersets will ever lead to an NNR edit rule that is previously not found, again reducing the total number of combinations to test. In other words, a contributing set leading to a potential NNR edit rule, should be minimal in the sense that it should contain as few edit rules as possible. A consequence of this proposition is stated in the following.

Corollary 1 (Pruning rule 3)**.** *Each edit rule E^k in a contributing set $\mathcal{E}_c$ that leads to the generation of an essentially new edit rule E^* by means of generator g, should have at least one value $v \in D_g$ in A_g^k that is not included in any of the $A_g^{k'}$ of each $E^{k'} \in \mathcal{E}_c \setminus E^k$. Otherwise, E^* is redundant to the essentially new edit rule E^{**} that can be generated by means of contributing set $\mathcal{E}_{c'} = \mathcal{E}_c \setminus E^*$ and generator g.*

Corollary 1 states that, when extending a certain combination of edit rules with an edit rule E^k, A_g^k should contain at least one value of D_g that is not included in any of the value sets of g of the other edit rules in this combination. Otherwise, any contributing set $\mathcal{E}_c$ containing the extended combination and leading to an essentially new edit rule E^* will not be minimal, as E^k can be removed from $\mathcal{E}_c$. All edit rules that have this property in a certain set of edit rules $\mathcal{E}$ will be called *contributing edits in $\mathcal{E}$* in the remainder.

4.2. Improving nominal edit rule implication

After introducing the pruning rules in 4.1, it is possible to propose an improved implication algorithm for the retrieval of (potential) NNR edit rules by means of generator g, exploiting these rules. Important to notice is that it is a *breadth-first* algorithm, meaning that it starts with maintaining singletons of edits, and keeps extending combinations until one of the stop conditions is reached. The reason for this is that the pruning rules all bear the property that once a combination is found that will never lead to an NNR edit rule, all supersets should not be tested anymore, so we can prune at that point. The pseudocode of this breadth-first algorithm is given in Algorithm 2.

As input parameters, the algorithm expects any set of edit rules $\mathcal{E}$ and a generator g used to generate all (potential) NNR edit rules from. These generated edits will be added to the result set $\mathcal{E}_{\text{NNR}}$, initialized on line 2. Then, on line 3, only the candidate contributors for generator g are selected from $\mathcal{E}$, as only these edit rules can be used to generate an NNR edit rule by means of generator g. On line 4-5, it is checked if every value of D_g appears in at least one edit rule in $\mathcal{E}_g$, otherwise, it is not possible to generate an NNR edit rule and the algorithm will return an empty result set. After the initial checks are done, the algorithm starts creating combinations of edit rules to test over $|D_g|$ different iterations. Each iteration i is responsible for a certain value $v_i \in D_g$, in the sense that each edit rule E^k with $v_i \in A_g^k$ is added to the combinations that remain after completing iteration $i-1$. Later, we will prove that there is an optimal order in which the values of D_g should be visited, but for now, we assume that this order is random. The combinations in the first iteration L_1 are singletons, one for each edit rule E^k with $v_1 \in A_g^k$ (line 6). For all subsequent iterations, the edits $E^k \in \mathcal{E}_{v_i}$ with $v_i \in A_g^k$, are selected from $\mathcal{E}_g$ (line 8) and each of these edit rules is added separately to each combination $\mathcal{E}_{c_{i-1}}$ of the previous iteration L_{i-1} (line 10-25), but only if all edits in this extended combination $\mathcal{E}_{c_i}$ remain to be contributing edits in $\mathcal{E}_{c_i}$, according to pruning rule 3 (line 16-17). For all newly formed combinations it is tested by applying Lemma 1 if they contribute to the generation of (1) a tautology (line 20), (2) an essentially new edit rule (line 22) or (3) a non-essentially new, implied edit rule (otherwise) by means of generator g. In the second case, the generated edit rule E^{**} is added to the result set (line 23). Because of pruning rule 1 and 2, only in the third case, the combination is added to L_i (line 25). However, if a combination of the previous iteration L_{i-1} resulted already in an implied edit rule E^* with $v_i \in A_g^*$, it is also added to L_i (line 12-14). Indeed, those combinations already

Algorithm 2 A nominal, breadth-first edit rule implication algorithm.

```
function getNNREditRules($\mathcal{E}$, $g$)
    $\mathcal{E}_{\text{NNR}} \leftarrow \emptyset$
    $\mathcal{E}_g \leftarrow [E^k \mid E^k \in \mathcal{E} \wedge g \in \mathcal{I}(E^k)]$
    if $\bigcup_{E^k \in \mathcal{E}_g} A_g^k \neq D_g$ then
        return $\emptyset$
    $L_1 \leftarrow \{\{E^k\} | E^k \in \mathcal{E}_g \wedge v_1 \in A_g^k\}$
    for $i \leftarrow 2$ to $|D_g|$ do
        $\mathcal{E}_{v_i} \leftarrow \{E^k | E^k \in \mathcal{E}_g \wedge v_i \in A_g^k\}$
        $L_i \leftarrow \{\}$
        for all $\mathcal{E}_{c_{i-1}} \in L_{i-1}$ do
            $E^* \leftarrow \text{FH}(g, \mathcal{E}_{c_{i-1}})$
            if $v_i \in A_g^*$ then
                $L_i \leftarrow L_i \cup \{\mathcal{E}_{c_{i-1}}\}$
                continue
            for all $E^k \in \mathcal{E}_{v_i}$ do
                if $A_g^k \subseteq A_g^* \vee A_g^* \subseteq A_g^k$ then
                    continue
                $\mathcal{E}_{c_i} \leftarrow \mathcal{E}_{c_{i-1}} \cup \{E^k\}$
                $E^{**} \leftarrow \text{FH}(g, \mathcal{E}_{c_i})$
                if $E^{**} = \emptyset$ then
                    continue
                if $g \notin \mathcal{I}(E^{**})$ then
                    $\mathcal{E}_{\text{NNR}} \leftarrow \mathcal{E}_{\text{NNR}} \cup \{E^{**}\}$
                else
                    $L_i \leftarrow L_i \cup \{\mathcal{E}_{c_i}\}$
    return $\mathcal{E}_{\text{NNR}}$
```

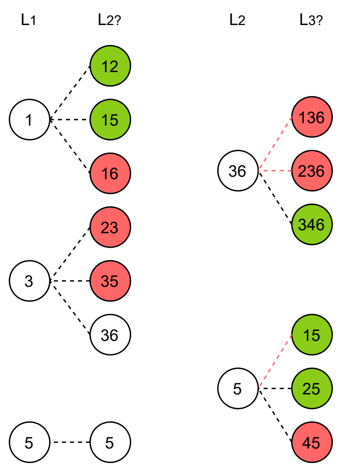


Figure 2: Visual representation of running the nominal implication algorithm given the edits from Table 1 as $\mathcal{E}$ and attribute `model` as g.

contain v_i and therefore, they can be passed automatically to the next iteration without the need for being extended. The algorithm will terminate after each value $v \in D_g$ is investigated and returns $\mathcal{E}_{\text{NNR}}$ upon completion.

To illustrate the execution of the algorithm, suppose that the edit rules, given in Table 1 are given as input $\mathcal{E}$ to the algorithm, together with generator $g = \texttt{model}$. A visual representation of the execution is given in Figure 2, in which each edit rule E^k is denoted by its index k. After the initial checks, all edit rules in $\mathcal{E}$ remain in $\mathcal{E}_g$, because attribute `model` is involved in each of the edit rules in $\mathcal{E}$ and all values $v \in D_{\texttt{model}}$ appear at least once. Suppose $v_1 = \text{P}$, which implies that L_1 consists of all singletons of the edits $E^k \in \mathcal{E}_g$ with $\text{P} \in A^k_{\texttt{model}}$, i.e. $L_1 = \{\{E^1\}, \{E^3\}, \{E^5\}\}$. After this, each edit rule E^k with $v_2 = \text{S} \in A^k_{\texttt{model}}$ is added to the singletons of L_1 to create the combinations listed under $L_{2?}$. The combinations resulting in an essentially new edit rule E^{**} (e.g. $\{E^1, E^2\}$) are colored green and added to $\mathcal{E}_{\text{NNR}}$ and the combinations resulting in a tautology (e.g. $\{E^1, E^6\}$) or containing a non-contributing edit (e.g. $\{E^3, E^5\}$) are colored red. All other combinations generate a non-essentially new, implied edit rule and are added to L_2. Important to notice is that E^5 is passed automatically to L_2 without being extended because $\{\text{P, S}\} \subseteq A^5_{\texttt{model}}$. Finally, all edit rules

E^k with $v_3 = \mathrm{X} \in A^k_{\texttt{model}}$ are added to the remaining combinations and tested correspondingly. A possible optimization that is previously not mentioned is to keep track of all combinations that are already pruned, at the expense of increasing memory requirements. Indeed, as can be seen in the example given in Figure 2, the combination $\{E^1, E^3, E^6\}$ should actually not be tested anymore, because it is already known that one of its subsets ($\{E^1, E^6\}$) will never lead to the generation of an NNR edit rule. All combinations that could be pruned because of this reason are therefore presented by a red dashed line.

Now, it is possible to assert the following proposition.

Proposition 3. *Algorithm 2 generates all (potential) NNR edit rules given a set of nominal edit rules $\mathcal{E}$ and generator g.*

Proof. We restrict ourselves to a sketch of the proof. During the course of Algorithm 2, all generated essentially new edit rules are adopted in the result set, once encountered. Thereby, each combination containing at least one edit rule E^k with $v \in A^k_g$, for each value $v \in D_g$, is tested, as dictated by Definition 2, unless the combination does not (and will never) generate a (potential) NNR edit rule according to the pruning rules proposed in 4.1. $\square$

In the previous, we mentioned that Algorithm 2 generates all *potential* NNR edits. Indeed, it is sure that all essentially new edits are generated, but it is still possible that some edits will be redundant to others generated by Algorithm 2 or in another step of the FCF algorithm. To resolve this, it is possible to test each generated edit for being redundant to all other edits each time the implication algorithm terminates and treat redundant edits properly as described in [8].

To finish this section, the complexity of the algorithm is analyzed. A worst-case scenario is one in which (1) for each candidate contributor E^k, $|A^k_g| = 1$, (2) for each $v \in D_g$, n edits exist with v in their value set of g, and (3) no combinations can be pruned. In this case,

$$\sum_{i=2}^{|D_g|} n^i = \sum_{i=0}^{|D_g|} n^i - n^1 - n^0 = \frac{n^{|D_g|+1} - 1}{n - 1} - n - 1 \qquad (3)$$

combinations have to be tested by Algorithm 2, according to the closed-form formula for the geometric series. Although this formula is still exponential in terms of $|D_g|$, it will always be lower than the upper bound given in Eq. 2, considering there are $n * |D_g|$ candidate contributors for g. Further, if n_i edit rules exist with v_i in their value set of g with n_i not necessarily the same for each value $v_i \in D_g$ and the other assumptions remain, the number of combinations to test in a worst-case scenario equals

$$\sum_{i=2}^{|D_g|} \prod_{j=1}^{i} n_j = (n_1 * n_2) + (n_1 * n_2 * n_3) + \ldots + (n_1 * \ldots * n_{|D_g|}). \tag{4}$$

To minimize the result of Eq. 4, one can sort the values $\{n_1, \ldots, n_{|D_g|}\}$ in increasing order, as stated by the following proposition.

Proposition 4. *Given a set of strict positive integers* $\{n_1, \ldots, n_{|D_g|}\}$. *The result of Eq. 4 will be minimal if the numbers are passed in increasing order.*

Proof. Suppose for some k and l, with $k < l \leq |D_g|$ that $n_k > n_l$. Now, Eq. 4 can be written as

$$S = \sum_{i=2}^{k-1} \prod_{j=1}^{i} n_j + \sum_{i=k}^{l-1} \prod_{j=1}^{i} n_j + \sum_{i=l}^{|D_g|} \prod_{j=1}^{i} n_j = T_1 + n_k * T_2 + n_k * n_l * T_3$$

If we swap n_k and n_l, the sum becomes $S' = T_1 + n_l * T_2 + n_k * n_l * T_3$. This gives $S' - S = (n_l - n_k) * T_2 < 0$ because $n_l < n_k$, which implies that swapping a larger positive integer to a later position in the sequence, will decrease S. Therefore, S will be minimal if the numbers are passed in increasing order. $\square$

Proposition 4 states that it is good practice to start with investigating the value of D_g that appears least in the value sets of the generator of the candidate contributors, and continue increasingly. A more empirical evaluation of the algorithm and the proposed pruning rules is presented in Section 7.

5. Edit rules for ordinal data

In this section, edit rules for ordinal data are introduced and defined formally. In order to do this, we propose concepts, notations and definitions related to intervals on totally ordered sets in 5.1. Next, ordinal edit rules are defined in 5.2 and in 5.3, an algorithm for ordinal edit rule implication is proposed.

5.1. Intervals on totally ordered sets

As introduced in the following, the notion of ordinal edit rules is built on the concept of intervals [10, 45]. In mathematics, intervals are usually defined as convex subsets of the real numbers, but here, we want to use the concept of an interval on countable sets as well. Therefore, we provide some notations and definitions of this more general notion of intervals. We consider attributes $\mathcal{R} = \{1, \dots, m\}$ where the domain D_j of attribute j is equipped with a *total* order $\leq_j$. In case j is understood, we denote $\leq_j$ simply by $\leq$. The set D together with the total order $\leq$ is often denoted by $(D, \leq)$ and is called a *chain* [30].

Definition 4 (Interval on a chain)**.** *An interval $I = [\ell, u]$ on a chain $(D, \leq)$ is a subset of D determined by two values $\ell \in D$ (the lower bound) and $u \in D$ (the upper bound) such that $\ell \leq u$ and is defined by $I = \{v \mid v \in D \wedge \ell \leq v \leq u\}$.*

For the sake of simplifying our notations, we assume that all D_j are *finite*. This assumption allows us to consider *closed* intervals only and erases the necessity to introduce the strict variant of the order $\leq$. Moreover, if D_j is finite, it always contains unique minimal and maximal elements, which we denote respectively by $\underline{v}_j$ and $\overline{v}_j$. We now introduce some convenient definitions.

Definition 5 (Interval equivalence)**.** *Two closed intervals $I' = [\ell', u']$ and $I'' = [\ell'', u'']$ on a chain $(D, \leq)$ are called* left-equivalent *(denoted by $I' \equiv_\ell I''$) if $\ell' = \ell''$ and* right-equivalent *(denoted by $I' \equiv_u I''$) if $u' = u''$.*

Definition 6 (Interval connection)**.** *Two closed intervals $I' = [\ell', u']$ and $I'' = [\ell'', u'']$ on a chain $(D, \leq)$ are* connected *(denoted by $I' \leftrightarrow I''$) if the union of I' and I'' (defined by $\{v \mid v \in D \wedge (v \in I' \vee v \in I'')\}$) is also an interval on $(D, \leq)$.*

On a countable chain $(D, \leq)$, two intervals are connected if either their intersection is not empty (e.g. $[1, 3]$ and $[2, 4]$), or the lower bound of one interval is the next element in the chain of the upper bound of the other interval (e.g.

$[1,2]$ and $[3,4]$). Finally, we consider two partial orders $\preceq_\ell$ and $\preceq_u$ on the set of intervals for a chain $(D, \leq)$.

Definition 7 (Interval partial orders)**.** *For two closed intervals $I' = [\ell', u']$ and $I'' = [\ell'', u'']$ on a chain $(D, \leq)$, we have that $I' \preceq_\ell I''$ if $\ell' \leq \ell''$ and $I' \preceq_u I''$ if $u' \leq u''$.*

5.2. Ordinal edit rules

With the definitions and notations set, we introduce an *ordinal edit rule.*

Definition 8 (Ordinal edit rule)**.** *An ordinal edit rule E^k over a set of m attributes $\mathcal{R} = \{1, \ldots, m\}$ with respective totally ordered domains $D_1, \ldots, D_m$ covers a subset $P(E^k)$ of the universe $D_1 \times \ldots \times D_m$, which is defined by*

$$P(E^k) = \prod_{j=1}^{m} I_j^k \tag{5}$$

with I_j^k an interval on the chain $(D_j, \leq)$.

An example of an ordinal edit rule based on Figure 1 is $A : [2, 8] \times B : [1, 4]$.

It can be seen that the definition of ordinal edit rules is closely related to Definition 1, with sets replaced by intervals. From this point of view, ordinal edit rules are a natural extension of nominal edit rules for ordered data in particular (which includes data measured on the ordinal, interval and ratio scale as well). Indeed, intervals are sets that can be concisely described in terms of a lower and upper bound. However, not every subset from D_j can be written as an interval. In the strict sense, this is not an obstacle with respect to expressiveness as it trivially holds that any subset of $D_1 \times \ldots \times D_m$ can be expressed by a set of ordinal edit rules. Of course, some representations will contain many edit rules and if we extend our reasoning to the case of infinite domains, then some representations will contain infinitely many edit rules. Additionally, when one imposes an artificial order on the values of each attribute, these observations imply that each nominal edit rule can be written as a set of ordinal edit rules. Vice versa, it is possible to write each ordinal edit rule as one nominal edit rule. Therefore, the aforementioned properties of nominal edit rules still hold.

5.3. Ordinal edit rule implication

Now, we will show that the properties of intervals on a chain $(D, \leq)$ can be exploited during ordinal edit rule implication. Before we propose an algorithm for this, we must first point out that the representation of ordinal edit rules is not closed under edit rule implication. Indeed, according to Lemma 1, edit rule implication will introduce the union of intervals for the generating attribute and the union of two intervals is not necessarily an interval itself. However, for an essentially new edit rule, the generating attribute is no longer involved (Definition 2). Moreover, for non-generating attributes, an implied edit rule features the intersection of intervals, which is ensured to be an interval itself. As such, our representation of ordinal edit rules is closed under edit rule implication of essentially new edits. Because we are only interested in edit rules that are essentially new, this weaker form of closure is sufficient.

A first step is to check the set $\mathcal{E}_g \subseteq \mathcal{E}$, containing all candidate contributors for generator g, with respect to the intervals for g. If each such interval is either left-equivalent or right-equivalent with D_g, then any NNR edit will be generated by using a contributing set consisting of *exactly* two edit rules. This result is formalised in the following proposition.

Proposition 5. *Let $\mathcal{E}_g$ be a set of ordinal edit rules over a set of m attributes $\mathcal{R} = \{1, \ldots, m\}$ with $\forall E^k \in \mathcal{E}_g$, $g \in \mathcal{I}(E^k)$. If $\forall E^k \in \mathcal{E}_g : I^k_g \equiv_\ell D_g \vee I^k_g \equiv_u D_g$, then any contributing set $\mathcal{E}_c \subseteq \mathcal{E}_g$ used for generating an NNR edit rule E^* by means of generator g, contains exactly two edit rules.*

Proof. We can partition $\mathcal{E}_g$ into $\mathcal{E}_{g^\ell}$ and $\mathcal{E}_{g^u}$, where $\mathcal{E}_{g^\ell}$ contains all edits that are left-equivalent with D_g and $\mathcal{E}_{g^u}$ contains all edits that are right-equivalent with D_g. This is a partition because any $E \in \mathcal{E}_g$ is by definition in at least one of both sets $\mathcal{E}_{g^\ell}$ and $\mathcal{E}_{g^u}$ and moreover, because g enters in each $E \in \mathcal{E}_g$, an edit cannot be in both sets. Because of the involvement assumption, no sets containing edits of $\mathcal{E}_{g^\ell}$ or edits of $\mathcal{E}_{g^u}$ only can ever generate an essentially new edit. Hence, we must consider at least one edit rule $E^\ell \in \mathcal{E}_{g^\ell}$ and one edit rule $E^u \in \mathcal{E}_{g^u}$. Now, we can see that for two edits E^{ℓ_1} and E^{ℓ_2} from $\mathcal{E}^\ell_g$ we always have either $I_g^{\ell_1} \subseteq I_g^{\ell_2}$ or $I_g^{\ell_2} \subseteq I_g^{\ell_1}$. Because of Corollary 1, we know that any contributing set containing E^{ℓ_1} and E^{ℓ_2} will not be minimal and either E^{ℓ_1} or E^{ℓ_2} can be removed. The same holds for edits from $\mathcal{E}_{g^u}$. $\square$

The consequence of Proposition 5 is that, if the premise is satisfied, we can generate all NNR edit rules in time $\mathcal{O}\left(|\mathcal{E}_g|^2\right)$. Of course, the requirements of Proposition 5 are quite strong and it is likely that these requirements will not be met. In that case, we resort to a more general strategy for generating all ordinal NNR edit rules. In this work, we propose a stack-based ordinal edit rule implication algorithm as such a more general strategy.

As can be seen in the following, the algorithm will exploit the same properties as introduced in 4.1 in the case of nominal edit rules. The main idea of the algorithm is that validating these properties for ordinal edit rules can be done more efficiently by using the following observations and the fact that each value set (interval) is represented by a lower and upper bound only, instead of being represented by an enumeration of set elements in the case of nominal edit rules.

Proposition 6. *Consider a set of ordinal edit rules $\mathcal{E}$ over m attributes $\mathcal{R} = \{1, \ldots, m\}$, with $\mathcal{E}_g \subseteq \mathcal{E}$ the set of candidate contributors for generator $g \in \mathcal{R}$. Suppose also that E^* is an implied edit rule generated by means of a contributing set $\mathcal{E}_c \subseteq \mathcal{E}_g$ and generator g. Let L be the list of intervals for attribute g that appear in edit rules from $\mathcal{E}_c$ sorted according to $\preceq_\ell$, then E^* is not NNR if any of the following conditions are met.*

(a) $\exists i : \neg\left(L[i] \leftrightarrow L[i+1]\right)$

(b) $\exists i : L[i] \equiv_\ell L[i+1]$

(c) $\exists i : L[i+1] \preceq_u L[i]$

(d) $\exists i : L[i-1] \leftrightarrow L[i+1]$

Proof. $\boxed{a}$ If two consecutive intervals in the sorted list do not connect, then the union of all intervals is not equal to D_g and E^* is not essentially new, according to Definition 2.

$\boxed{b}$ If there are two intervals with an equal left bound, then one interval is a subset of the other. According to Corollary 1 (pruning rule 3), if E^* is essentially new, the contributing set leading to the generation of E^* is not minimal as the edit rule that corresponds with the smallest interval can be removed.

$\boxed{c}$ If the upper bound of $L[i+1]$ is smaller than or equal to the upper bound of $L[i]$, then $L[i+1] \subseteq L[i]$, because L is sorted by $\preceq_\ell$. According to Corollary 1 (pruning rule 3), if E^* is essentially new, the contributing set leading to the generation of E^* is not minimal as the edit rule that corresponds with $L[i+1]$ can be removed.

$\boxed{d}$ If two non-consecutive intervals $L[i-1]$ and $L[i+1]$ connect, we can skip the interval at position i. According to Proposition 2 (pruning rule 2), if E^* is essentially new, the contributing set leading to the generation of E^* is not minimal as the edit rule that corresponds with $L[i]$ can be removed. $\square$

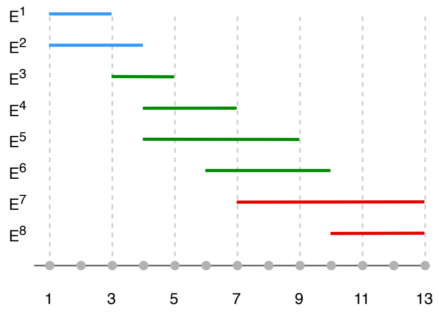


Figure 3: An example scenario of intervals that appear in edit rules for the generating attribute g. The intervals cover the entire domain D_g and are sorted according to $\preceq_\ell$.

Algorithm 3 presents the pseudocode of an ordinal edit implication algorithm, given a set of ordinal edit rules $\mathcal{E}$ and generator g. In order to illustrate its main ideas, the situation sketched in Figure 3 will be considered. In this example, eight candidate contributors for generator g exist (i.e. $\mathcal{E}_g = \{E^1, \ldots, E^8\}$). For each edit rule $E^k \in \mathcal{E}_g$, I_g^k is shown in the diagram. The domain of generator g equals $D_g = \{1, \ldots, 13\}$.

The resulting NNR edit rules are collected in $\mathcal{E}_{\text{NNR}}$, which is initially empty (line 2). First, $\mathcal{E}_g$ is created (line 3) and it is checked if every value of D_g appears in at least one edit rule in $\mathcal{E}_g$ (line 4-5, cfr. Algorithm 2). The algorithm is constructed around two data structures. The first is a *stack* $\mathbb{S}$ that keeps all sequences (or combinations) of edits that can still lead to a (potential) NNR edit rule. This stack is initialized with sequences consisting of single edits, where each edit is left-equivalent with D_g (line 6). In the example of Figure 3, the stack would be initialized with two single-edit sequences: one with edit E^1 and one with edit E^2 (blue intervals). The second data structure is a *sorted list* $\mathbb{L}$ that keeps all edits that are not initially added to the stack. The order in this list is determined by $\preceq_\ell$ (line 7). In the example of Figure 3, the list contains edits E^3 to E^8 in that order. In a final initialization step, for each value $v \in D_g$, the *first* index i in the sorted list $\mathbb{L}$ is determined such that the edit rule at $\mathbb{L}[i]$ has a left bound that is strictly greater than v. These indices are assigned to the variables idx_v (line 8). We note here that it suffices to calculate only those idx_v for which there is some edit in $\mathcal{E}_g$ that has v as the left bound of its interval for g. In Figure 3, there are eight edits in $\mathcal{E}_g$ and they correspond to six unique left bounds: 1, 3, 4, 6, 7 and 10, so we need to keep an index for only these values. In the example, the index is 4 for value 4 because the first edit (E^6) of which the left bound is strictly greater than 4 is located at position 4 in $\mathbb{L}$.

After the initialization phase, the algorithm starts popping sequences from the stack. If a sequence $\mathcal{E}_s$ is popped from the stack, the last edit of $\mathcal{E}_s$ is assigned to the variable E^s. Then sequence $\mathcal{E}_s$ is extended with edits from the sorted list. In order to do so, the *left bound* of interval I^s_g is considered and the index for that value is looked up. Adding edits to the sequence that occur *before* idx_v is of no use as our sequence of intervals should always progress from left to right in the sense of $\leq_g$ (Proposition 6). Also, idx_v will skip those edits for which the left bound of interval I^s_g is equal to the left bound of the last edit in the sequence (Proposition 6, (b)). For each edit in $\mathbb{L}$, the intervals should be connecting (line 14). If this is not the case, then by condition (a) from Proposition 6 we know that the edits are not essentially new. In fact, we can break the entire loop because due to the sorting of $\mathbb{L}$, all remaining edits in $\mathbb{L}$

Algorithm 3 An ordinal edit rule implication algorithm.

```
1: function getOrdinalNNREditRules($\mathcal{E}$, $g$)
2:     $\mathcal{E}_{\text{NNR}} \leftarrow \emptyset$
3:     $\mathcal{E}_g \leftarrow [E^k \mid E^k \in \mathcal{E} \wedge g \in \mathcal{I}(E^k)]$
4:     if $\bigcup_{E^k \in \mathcal{E}_g} I_g^k \neq D_g$ then
5:         return $\emptyset$
6:     $\mathbb{S} \leftarrow \{[E^k] \mid E^k \in \mathcal{E}_g \wedge I_g^k \equiv_\ell D_g\}$
7:     $\mathbb{L} \leftarrow$ sort $\left(\left[E^k \mid E^k \in \mathcal{E}_g \wedge \neg\left(I_g^k \equiv_\ell D_g\right)\right], \preceq_\ell\right)$
8:     $\forall v \in D_g : \text{idx}_v = \min\{i \mid v < \text{left}(\mathbb{L}[i])\}$
9:     while $\mathbb{S} \neq \emptyset$ do
10:        $\mathcal{E}_s \leftarrow$ pop $(\mathbb{S})$
11:        $E^s \leftarrow \mathcal{E}_s\,[|\mathcal{E}_s|]$
12:        for all $i \in \{\text{idx}_{\text{left}\left(I_g^s\right)}, \ldots, |\mathbb{L}|\}$ do
13:            $E^k \leftarrow \mathbb{L}[i]$
14:            if $\neg\left(I_g^s \leftrightarrow I_g^k\right)$ then
15:                break
16:            $E^{s'} \leftarrow \mathcal{E}_s\,[|\mathcal{E}_s - 1|]$
17:            if $I_g^k \subseteq I_g^s \vee \left(|\mathcal{E}_s| \geq 2 \wedge I_g^{s'} \leftrightarrow I_g^k\right)$ then
18:                continue
19:            $E^* \leftarrow \text{FH}(g, S \oplus E^k)$
20:            if $E^* = \emptyset$ then
21:                continue
22:            if $g \notin \mathcal{I}(E^*)$ then
23:                $\mathcal{E}_{\text{NNR}} \leftarrow \mathcal{E}_{\text{NNR}} \cup \{E^*\}$
24:            else
25:                push$\left(\mathbb{S}, S \oplus E^k\right)$
26:    return $\mathcal{E}_{\text{NNR}}$
```

will fail this connection test. Next, conditions (c) and (d) from Proposition 6 are verified and if any of them is met, the loop is continued (line 17-18). If none of the above pruning rules can be applied, the implied edit rule is computed with the extended sequence (line 19). If it is a tautology (line 20), the algorithm continues. If it is an essentially new rule, it is added to $\mathcal{E}_{\text{NNR}}$ (line 23). Else, the extended sequence of edits is pushed to the stack (line 25). This process continues until the stack is empty and finally, all generated NNR edit rules are returned.

We can now assert the following proposition.

Proposition 7. *Algorithm 3 generates all (potential) NNR edit rules given a set of ordinal edit rules $\mathcal{E}$ and generator g.*

Proof. We restrict ourselves again to a sketch of the proof. Clearly, for any NNR edit rule, the contributing set requires at least one edit E^k for which the interval I_g^k is left-equivalent to D_g. Else, the edit rule can never be essentially new. Initially, all edits E^k for which I_g^k is left-equivalent to D_g are pushed on the stack. From there, all possible combinations of edit rules are tested, unless the combination does not (and will never) generate a (potential) NNR edit rule according to Proposition 6 (based on the pruning rules given in 4.1). $\square$

The theoretical time-complexity of the algorithm can be analyzed in the same way as done for Algorithm 2 given in 4.2. The benefit of validating the pruning rules for ordinal data compared to nominal data is that ordinal edit rules use a representation of intervals only defined by a lower and an upper bound, instead of being represented by an enumeration of set elements in the case of nominal edit rules. This implies that comparing value sets in the case of ordinal edit rules comes down to comparing two values instead of a set of values in the case of nominal edit rules and the larger the intervals of the generator are, the more beneficial for the performance of edit rule implication. Also, the properties of intervals can be used to extend combinations in a more intelligent way. A more empirical evaluation of the algorithm and the proposed pruning rules is presented in Section 7.

To conclude this section, two additional observations are given. First, as already mentioned in 5.2, it is possible to convert a set of nominal edit rules to ordinal edit rules and vice versa. Therefore, Algorithm 2 and 3 can both be used for edit rule implication, without making any assumption on the type of data. Additionally, the findings concerning edit folding, described in Section 3, can be used given either an explicit set of nominal or ordinal edit rules. Besides that, we point out that Algorithm 3 requires some modifications if D_g is no longer finite. First, if D_g becomes countably infinite, it is crucial to store indices for only those values that appear as a left bound in the candidate contributors $\mathcal{E}_g$. Second, if D_g is not a countable set (e.g., $\mathbb{R}$), then we must introduce *open intervals* to keep the same expressiveness of ordinal edit rules. In that case, the computation of indices needs to account for open intervals. Moreover, in the case of uncountable domains, intervals are connecting as soon as they are not disjunct. Studying this in more detail is subject to future work.

6. Related work

For more than half a century, the problem of automatic repair of (statistical) data inconsistencies by means of data quality rules has been investigated thoroughly [19, 29, 38, 39]. Fellegi and Holt proposed in their seminal paper [23] that a possible way to solve this problem is to tackle it as a two-step process, consisting of error localization (determining the attributes in error) on the one hand and imputation (estimating correct values for the attributes in error) on the other hand. Besides analyzing this problem, Fellegi and Holt also introduced an essential concept related to this problem, which they called an edit rule [23]. Although edit rules belong to the category of tuple-level constraints and can only represent forbidden data objects by means of equalities of attribute values, it is already stated in Section 1 and Section 2 that exploiting the properties of edit rules could simplify the first step in the process of repairing data inconcistencies tremendously. Because our work focuses especially on error localization by means of edit rules, an overview of the most notable literature related to this topic is given in the remainder of this section, together with a positioning of our work among these contributions.

The techniques to locate data inconsistency errors in data objects can be subdivided into two categories. A first category captures the techniques that treat this problem as (a variant of) the covering set problem. Fellegi and Holt stated that this can only be done correctly if all necessary information is made available by means of edit rule implication which relies on the techniques proposed in their framework [23]. Although this method was proven to be useful, generating all necessary edit rules, captured in a sufficient set, could become labour-intensive. Therefore many researchers, such as Liepins [32, 33, 34], Garfinkel et al. [25], Winkler [46, 47], Boskovitz [8] and Chen [11, 13], proposed and investigated algorithms to deal with this efficiently and correctly. Apart from the techniques that generate an entire sufficient set, techniques exist in which this generation is not fully completed, but each failing data object is treated separately and only the edits that are required to solve the problem by means of the covering set method for the object under investigation are generated by means of edit implication. However, a disadvantage of these techniques is that when the data set is updated, one has to reconsider the problem. Contributions exploiting this method are proposed by Garfinkel et al. [25] and by Chen and Winkler [13]. Our work is a continuation of the above-mentioned contributions in the sense that it investigates and improves the remaining shortcomings of the current techniques used for edit rule implication. In particular, in Section 3, we show that different edit rule sets that capture the same set of inconsistencies, can result in different performances. Besides that, in Section 4, we investigate efficient ways of evaluating pruning strategies exploited by the nominal implication algorithm.

A second category includes approaches that do not use (any variant of) the covering set method. Examples include vertex generation methods [16, 18, 44] based on the Chernikova algorithm [14], branch-and-bound strategies [16, 17, 20] or treating it as a dynamic disjunctive facet problem [16].

Besides the techniques used to solve the error localization problem on its own, the entire repair problem (including error localization and imputation) can also be solved at once [19, 31, 37]. An important contribution exploiting this is due to Bankier et al. [2] as they used a nearest-neighbour-based approach to identify, for each failing object, a donor object that resembles the failing object as close as possible. Although it is noted that Nearest-Neighbour Imputation (NIM), works well for particular applications (e.g. census editing), the covering set-based approaches described above are more generally applicable and can be used in situations where NIM would not work at all [12, 19].

It should be noted that most of the approaches mentioned above can also be applied to continuous or mixed data, yet in a slightly different form [17, 19, 20, 26, 31]. Although, no particular attention is given to implication of edit rules for ordered data, which tends to be useful, as explained in Section 1 and Section 5, and shown to be more efficient than nominal implication.

To finish this discussion, two frameworks are mentioned that are available to repair data inconsistencies against a set of rules, called HoloClean [43] and Llunatic [27]. An extensive overview of state-of-the-art research in the field of (statistical) data cleaning can be found in [19].

7. Evaluation

In this section, the practicality of the algorithms and techniques described in Section 3-5 is investigated thoroughly by using them in a set of experiments. During the evaluation, we try to provide an answer to the following questions.

- What is the impact of edit folding on the total number of edit rules and entering attributes in an explicit set of edit rules?
- What is the impact of edit folding on the performance of the FCF algorithm?
- What is the impact of using the nominal implication algorithm (Algorithm 2) and the ordinal implication algorithm (Algorithm 3) on the performance of the FCF algorithm?
- What is the impact of using the pruning rules in the edit implication algorithms (Algorithm 2 and Algorithm 3) on the number of combinations to test?

With these research questions in mind, we performed a number of experiments described below. The results listed in this section are obtained by using custom implementations of the algorithms in Java (version 8) executed on an Intel Core i7-8550U processor (1.8GHz) with 16GB of RAM running Windows 10. To test the performance of the FCF algorithm, we used a custom implementation proposed by Boskovitz [8], adopting different edit rule implication algorithms (including our proposals) for executing step 2.

7.1. Data and edit set characteristics

Table 2: Overview of the main characteristics of the data and edit sets used in the experiments.

data set	$\|R\|$	$\|\mathcal{R}\|$	$\|\mathcal{D}\|$	value type	edit set id.	$\|\mathcal{E}\|$	$\sum \|\mathcal{I}(E)\|$
Adult	48842	11	202	categorical	A1	146	328
					A2	205	485
Mushroom	8124	23	119	categorical	M1	206	665
					M2	299	996
Breast Cancer	699	9	90	integer	BC1	57	141
Trials	1512	25	95	categorical	T1	209	518

During the experiments reported in the following, six (explicit) edit sets based on four different data sets are used. An overview of the main characteristics of the data sets, including the number of data objects $|R|$, the number of attributes $|\mathcal{R}|$, the total number of values $|\mathcal{D}|$ and the type of the attribute values, is given in Table 2. Also, Table 2 gives an overview of the main characteristics of the six edit sets, including the size of the edit set $|\mathcal{E}|$ and the total number of entering attributes $\sum |\mathcal{I}(E)|$ over all edits in the set in particular. Note that all edit sets also have a unique identifier in order to refer to the sets easily.

Three data sets (Adult, Mushroom, Breast Cancer), commonly used as a standard when evaluating research on data consistency [5, 19, 41] can be downloaded from the UCI Machine Learning repository[4]. The data in Adult is extracted from the 1994 US Census database and relates demographic features to income of US citizens. For the experiments, it is preprocessed the same way as described in [41] by discretizing ages and removing other continuous attributes. The Mushroom data set contains an overview of the physical characteristics of mushrooms extracted from The Audubon Society Field Guide to North American Mushrooms [35]. The Breast Cancer data set originates from the Breast Cancer Wisconsin Database and describes different types of breast cancer tumors. In this data set, only the integer, non-ID attributes are kept. Finally, the Trials data set contains real-life data, as it is (self-)composed by collecting data concerning the design of clinical trials reported in both the German trials register[5] and the US trials register[6]. Therefore, we can state that Trials will be used as a representative of a realistic dataset.

Concerning the edit sets, the Adult- and Mushroom-based edit sets contain edit rules (or more specifically, low lift forbidden itemsets that are transformed to edit rules) that are automatically discovered by using the FBIMiner algorithm, described in [41] with different values for the maximal lift threshold τ. The Breast Cancer-based edit set contains edit rules obtained by first constructing an isolation forest on the dataset and then convert small sized branches of the trees of that forest to edit rules [36]. Therefore, we do not have any notion about the quality of the edits in these sets. The Trials-based edit set, however, is self-composed and we tried, to the best of our knowledge, to capture as many correct, forbidden value combinations as possible in an edit rule form to meet the demand of having a real-life edit set on which characteristics we based the other edit sets as well. Moreover, based on the cardinality of this last set, we can report that edit sets containing more than 200 edit rules are no exception.

[4]`http://archive.ics.uci.edu/ml/`
[5]`http://drks.de`
[6]`http://clinicaltrials.gov`

Finally, it is noted that all edit rules in the given sets corresponding with a categorical data set have the property that each entering attribute in the edit rules consists of only one value[7]. We will call edit rules that have this form *single-value edits* in the remainder. The advantage of a nominal edit rule in this form is that it can be converted to exactly one ordinal edit rule, making the comparisons between nominal and ordinal edit rule algorithms more reasonable.

7.2. Impact of edit folding

Table 3: Performance results of the edit folding algorithm.

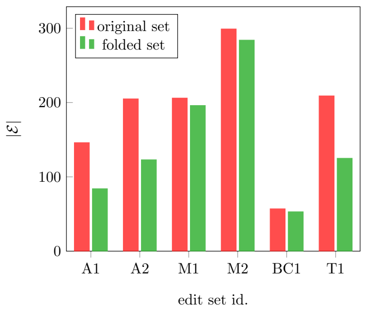
edit set id.	# recursive calls	worst case # recursive calls	mean execution time (in ms, based on 40 runs)
A1	8	146	14
A2	8	205	18
M1	5	206	31
M2	7	299	56
BC1	5	57	9
T1	9	209	41

In Section 3, a novel heuristic algorithm to reduce the total number of edits and entering attributes in an explicit set of edit rules is proposed, which exploits the properties of edit folding. Before evaluating the impact of edit folding on the number of edit rules and entering attributes, we show that the performance of the algorithm in practice tends to be much better than the theoretical worst case. Indeed, it can be seen from the results that are given in Table 3, that the number of times the algorithm is called recursively is far less than the worst-case scenario as well as n (with n the number of edit rules in the original set) for each given edit set and does not even reach the number of attributes m. This is

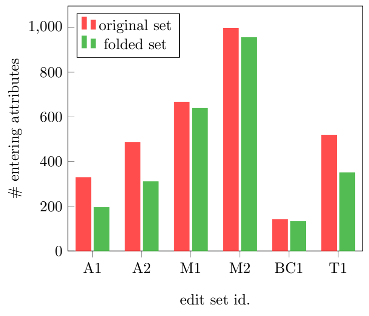
[7]Each edit rule set has the ability to be converted in such a way.

also something which could be expected, because only in very specific cases, an edit rule can be folded twice by means of the same folding attribute. Therefore, it can be stated that the mean execution time of the algorithm increases nearly linear in function of n, given a fixed number of attributes m, and the practical time complexity is estimated to be $O(nm^2)$, typically with $m \ll n$, as the results show.

Next, the impact on the total number of edits and the number of entering attributes over all edits is investigated. These results are given in Figure 4. For the Adult- and Trials-based edit sets (A1, A2, T1), the reduction in number of edits and entering attributes is tremendous. In these cases, the cardinality of the explicit set is reduced by at least 40% and the total number of entering attributes is reduced by at least 30% when using edit folding. The difference is much smaller for the Mushroom- and Breast Cancer-based edit sets (M1, M2, BC1) with a reduction of at most 8% for the number of edits and 6% for the total number of entering attributes. The reason for this is that these data sets consist of more than twice as many attributes as the Adult and Trials data sets, which reduces the probability that the condition holds, which states that edit rules should have equal values for $m-1$ attributes to fold.



(a) Number of edit rules.



(b) Number of entering attributes.

Figure 4: Impact of edit folding on the total number of edits and entering attributes.

To end this discussion on edit folding, it should be noted that a brute force approach for edit folding was tested on the edit set A1. Comparing these results to the results obtained by using the heuristic given above, it is noted that the folded set is exactly the same, but the mean execution time increased with a factor 13000. As a conclusion, it can be stated that the performance of the heuristic is very promising, but the reduction in number of edits and entering attributes highly depends on the data and edit sets. Therefore, it is necessary to investigate if it is always worth to apply edit folding, even if the reduction is limited. For this, the mean execution time of edit folding is compared to the total execution time of the FCF algorithm (both with and without first applying edit folding) in 7.3.

7.3. Efficiency of edit rule implication

Table 4: Mean execution times (in ms, based on 40 runs) of the FCF algorithm including different edit rule implication algorithms. The lowest mean execution time in the nominal case is underlined and the lowest mean execution time in general is indicated in bold.

edit set id.	Chen unfolded	breadth-first unfolded	Chen folded	breadth-first folded	ordinal unfolded
A1	56	21	32	20	**17**
A2	44538	1075	404	**82**	191
M1	1355	1251	1135	**1130**	1164
M2	4583	4412	4172	4154	**4027**
BC1	351	324	300	271	**175**
T1	5093	2558	2667	1331	**1046**

Nominal data. In the following, four different configurations of the FCF algorithm for nominal edit rules are compared to assess the performance of the proposed edit rule implication algorithm for nominal data. Each configuration differs either in the edit rule implication algorithm used during step 2 of the FCF algorithm or in the fact that the explicit set of edits is folded before it is given as input to the FCF algorithm. As can be seen from Table 4, the

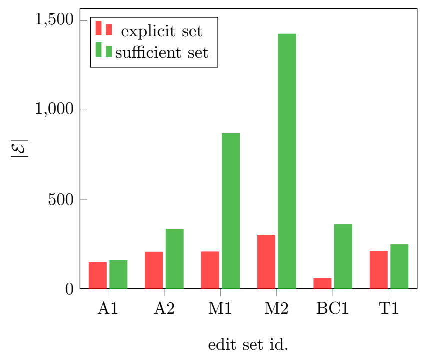


Figure 5: Impact of executing the FCF algorithm on the total number of edit rules, starting from an unfolded set of nominal edits.

proposed breadth-first algorithm (Algorithm 2) is compared to the Chen algorithm [11, 13], which is, to the best of our knowledge, state-of-the-art for this task. It must be said that the starting point of the Chen algorithm is slightly different, as it only searches for combinations of edit rules that together cover the entire domain of the generator, without testing if the resulting implied edit, is eventually NNR. Also, the breadth-first algorithm includes the optimization that already pruned combinations are kept and each newly formed combination that is a superset of one of the already pruned combinations is pruned itself. The mean execution times[8] over 40 runs are listed in Table 4, with the lowest mean for each edit set in the nominal case being underlined. Figure 5 shows the number of edits in each unfolded explicit set compared to the number of edits in its corresponding sufficient set in order to give an impression of the work that has to be done. The same factor of increase can be reported when starting from folded explicit sets, so no additional chart is given.

It is easy to notice that for each edit set, the mean execution times of the FCF algorithm including the breadth-first algorithm are lower than the mean execution times of the FCF algorithm including the Chen algorithm. Especially

[8]The mean execution times of the configurations with edit folding also include the time to fold the explicit set.

for edit set A2, there is a huge gain in performance when starting from an unfolded set. This is due to the fact that, when using the Chen algorithm, there is one branch in the Chen algorithm that tests many combinations and that is pruned early when using the breadth-first algorithm. For both edit sets M1 and M2, the mean execution times are comparable, which has to do with the limited depth of the branches. Therefore, we can say that the performance gain does depend on the fact if and when certain pruning rules can be applied (see 7.4). Besides that, if the mean execution times of edit folding, given in Table 3, are reconsidered, it can be stated that the execution time spend on edit folding is only a small fraction of the total execution time of the entire FCF algorithm (including edit folding). This implies that it is definitely worth to apply edit folding before generating a sufficient set, because for both algorithms and each edit set, the FCF algorithm given a folded set outperforms the FCF algorithm given an unfolded set. Moreover, as is argumented in Section 3, edit folding especially makes an impactful difference on the performance if it reduces the total number of edits and entering attributes vigorously, which is the case when applying it to edit sets A1, A2 and T1.

Ordinal data. To assess the performance of the ordinal implication algorithm, the mean execution times of the FCF algorithm including the ordinal implication algorithm (Algorithm 3) are also given in Table 4. It should be noted that the configuration to test the performance in the ordinal case does not include edit folding. The reason for this is that, when edit folding is applied, there is no guarantee that nominal edit rules can be converted to exactly one ordinal edit rule, which makes comparing mean execution times unreasonable. Also, the mean execution times do include the time for converting a nominal to an ordinal edit set, except for edit set BC1, which already consists of ordinal edit rules by definition and the optimization proposed in Proposition 5 is not considered.

From Table 4, it can be learned that for all edit sets, the ordinal FCF algorithm performs better than the nominal FCF algorithm, starting with an unfolded set of edit rules. For edit sets A2, BC1 and T1, there is a huge reduction in mean execution times by 82%, 46% and 59% respectively. Thereby, it can be seen from the lowest mean execution times indicated in bold that for edit sets A1, M2, BC1 and T1, the ordinal case even outperforms the nominal case starting from a folded set of edit rules, although slighlty. Therefore, it is highly recommended to consider converting a set of nominal edit rules to ordinal edit rules and using the ordinal edit rule implication algorithm, or at least, to use the nominal breadth-first algorithm preceded by edit folding, when the alternative is not possible.

7.4. Impact of pruning rules

In a final experiment, the impact of the pruning rules on the number of tested combinations during the FCF algorithm is evaluated to validate their effectiveness. Therefore, each combination that is tested to see if it leads to the generation of an NNR edit rule during the entire execution of the FCF algorithm is classified into one of the following categories: the combination should not be extended further (1) due to pruning rule 1, (2) due to pruning rule 2, (3) due to pruning rule 3, (4) because it is a superset of an already pruned combination or (5) the combination should be extended further because it leads to a non-essentially new, implied edit rule.

Table 5: Total number of tested combinations during the execution of the FCF algorithm starting from an unfolded set (nominal vs. ordinal case).

edit set id.	nominal	ordinal
A1	1045	1040
A2	22557	16878
M1	31510	31245
M2	81129	78662
BC1	31817	13542
T1	232152	198425

First, the total number of combinations tested during the entire execution of the nominal and ordinal FCF algorithm, starting from an unfolded set of edit rules, is given in Table 5. For all edit sets, the number of combinations tested in the ordinal case is lower than in the nominal case, which could explain part of the performance gain. The reason for this is that the addition of new edit rules to already existing combinations is considered in a more intelligent way than by the nominal breadth-first algorithm, reducing the total number of created combinations. Moreover, the total number of combinations tested is much lower than the theoretical upper bound defined in 4.2.

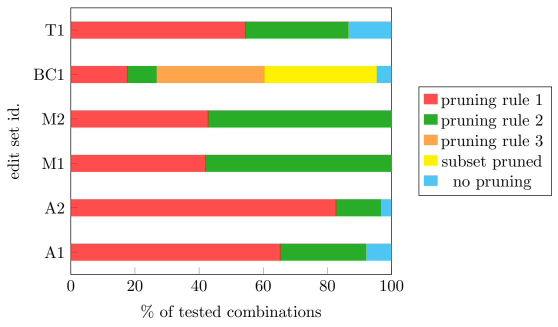


Figure 6: The percentage share of tested combinations during the entire nominal FCF algorithm assigned to each category.

In Figure 6, the percentage share of tested combinations during the entire FCF algorithm assigned to each category, defined above, is given. We only consider the nominal case, as the ordinal case is similar. In general, the impact of pruning rules 1 and 2 is huge. This is in fact very interesting, because pruning rule 1 is something that could not be used by the Chen algorithm as this algorithm does not generate implied edit rules during its execution. Indeed, this implies that validating pruning rule 1 comes at the performance cost of generating implied edits, but as explained in 7.3, this performance cost does not balance out the benefit of pruning. Besides that, there is only one edit set (BC1) that prunes combinations due to pruning rule 3 or because these combinations are supersets of already pruned combinations. For pruning rule 3, the explanation is twofold. First, the sets of edit rules (except for BC1) remained by the FCF algorithm only contains single-value edit rules and this will not change as only essentially new edits will be adopted. Second, because of the single-valuedness, edit rules that are considered during a certain iteration, will always contain one value, which is new compared to edit rules that are added to combinations in previous iterations. Therefore it can be said that, when one is not restricted to single-value edits, pruning rule 3 could be useful. For supersets of pruned combinations, the explanation is more difficult. However, for edit sets M1 and M2, notice that there are no combinations resulting in a non-essentially new implied edit rule, which means that all combinations are pruned at iteration 2. This makes it impossible for certain combinations to be assigned to become supersets of already pruned combinations. Therefore, it can be concluded that this strategy will be less effective in general when many combinations are pruned early, which will be often the case.

8. Conclusion

The main contribution of this work is to investigate and enhance the process of edit rule implication. Besides that, we provide a new way to deal with inconsistencies in ordinal data. The reason behind this is that, first, edit rule implication is an essential part in many data inconsistency repair strategies and second, it can become a complex process when edit rules are used on data measured beyond the nominal scale. As an overall result, we developed an efficient method to locate inconsistency errors. Indeed, as the evaluation of the methods shows, both implication algorithms, with the ordinal one as the frontrunner, outperform state-of-the-art methods. Moreover, the proposed methods both can be used for nominal edit rules and ordinal edit rules as one can easily transform edit rules from one type to the other. This implies that all properties and algorithms introduced for nominal edit rules also apply for ordinal edit rules and vice versa and additionally, both edit rule implication algorithms can also be used for a mix of nominal and ordinal data. Also, an essential part in each edit rule implication algorithm is to start with a set of edit rules that is as small as possible, which can be achieved by folding edits together.

Still, some research questions remain related to these topics. As is the case for many data quality rules, an efficient strategy to automatically discover ordinal edits lacks. Besides that, we only have considered a simple strategy to convert nominal to ordinal edit rules, but most likely, more intelligent ways to achieve this can be developed, that will enhance ordinal edit rule implication even further. Finally, edit rules capture only specific types of tuple-level constraints, based on equality of constant values. Therefore, investigating implication algorithms for more general types of tuple-level constraints (e.g. constraints based on inequalities or containing variables) can be very interesting.

References

[1] Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases: The Logical Level*. (1st ed.). USA: Addison-Wesley Longman Publishing Co., Inc.

[2] Bankier, M., Fillion, J.-M., Luc, M., & Nadeau, C. (1994). Imputing numeric and qualitative variables simultaneously. In *Proc. of the Section on Survey Research Methods* (pp. 242–247). American Statistical Association.

[3] Batini, C., Cappiello, C., Francalanci, C., & Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM Computing Surveys*, *41*.

[4] Batini, C., & Scannapieco, M. (2006). *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Berlin, Heidelberg: Springer-Verlag.

[5] Boeckling, T., Bronselaer, A., & De Tré, G. (2019). Mining data quality rules based on T-dependence. In *Proc. of the 11th Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology (EUSFLAT 2019)* (pp. 184–191). Atlantis Press.

[6] Bohannon, P., Fan, W., Flaster, M., & Rastogi, R. (2005). A cost-based model and effective heuristic for repairing constraints by value modification. In *Proc. of the 2005 ACM SIGMOD International Conference on Management of Data* SIGMOD '05 (p. 143–154).

[7] Bohannon, P., Fan, W., Geerts, F., Jia, X., & Kementsietsidis, A. (2007). Conditional functional dependencies for data cleaning. *Proc. of the 2007 IEEE International Conference on Data Engineering*, (pp. 746–755).

[8] Boskovitz, A. (2008). *Data Editing and Logic: The Covering Set Method from the Perspective of Logic*. Ph.D. thesis Australian National University.

[9] Bronselaer, A., De Mol, R., & De Tré, G. (2017). A measure-theoretic foundation for data quality. *IEEE Trans. on Fuzzy Systems*, *26*, 627–639.

[10] Burkill, J. C. (1924). Functions of intervals. *Proc. of the London Mathematical Society*, *s2-22*, 275–310.

[11] Chen, B.-C. (1998). *Set Covering Algorithms in Edit Generation*. Technical Report U.S. Bureau of the Census.

[12] Chen, B.-C., Thibaudeau, Y., & Winkler, W. E. (2002). A comparison study of ACS if-then-else, NIM, and DISCRETE edit and imputation systems using ACS data. In *Proc. of the Section on Survey Research Methods* (pp. 461–466). American Statistical Association.

[13] Chen, B.-C., & Winkler, W. E. (2004). The cutting plane algorithm in the error localization problem. In *Proc. of the 2004 Joint Statistical Meetings*.

[14] Chernikova, N. V. (1964). Algorithm for finding a general formula for the non-negative solutions of a system of linear equations. *USSR Computational Mathematics and Mathematical Physics*, *4*, 151–158.

[15] Chu, X., Ilyas, I. F., & Papotti, P. (2013). Discovering denial constraints. *Proc. of the VLDB Endowment*, *6*, 1498–1509.

[16] De Waal, T. (2003). *Processing of Erroneous and Unsafe Data*. Ph.D. thesis Erasmus University Rotterdam.

[17] De Waal, T. (2005). Automatic error localisation for categorical, continuous and integer data. *Statistics and Operations Research Trans.*, *59*, 57–99.

[18] De Waal, T. (2005). Solving the error localization problem by means of vertex generation. *Survey Methodology*, *29*, 71–80.

[19] De Waal, T., Pannekoek, J., & Scholtus, S. (2011). *Handbook of Statistical Data Editing and Imputation*. John Wiley & Sons.

[20] De Waal, T., & Quéré, R. (2003). A fast and simple algorithm for automatic editing of mixed data. *Journal of Official Statistics*, *19*, 383–402.

[21] Fan, W., & Geerts, F. (2012). *Foundations of Data Quality Management*. Morgan & Claypool Publishers.

[22] Fan, W., Geerts, F., Jia, X., & Kementsietsidis, A. (2008). Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. on Database Systems*, *33*.

[23] Fellegi, I., & Holt, D. (1976). A systematic approach to automatic edit and imputation. *Journal of the American Statistical Association*, *71*, 17–35.

[24] Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences (1st ed.). W. H. Freeman.

[25] Garfinkel, R. S., Kunnathur, A. S., & Liepins, G. E. (1986). Optimal imputation of erroneous data: Categorical data, general edits. *Operations Research*, *34*, 744–751.

[26] Garfinkel, R. S., Kunnathur, A. S., & Liepins, G. E. (1988). Error localization for erroneous data: Continuous data, linear constraints. *SIAM Journal on Scientific and Statistical Computing 9*, (pp. 922–931).

[27] Geerts, F., Mecca, G., Papotti, P., & Santoro, D. (2019). Cleaning data with llunatic. *The VLDB Journal*, .

[28] Gross, J. L., & Yellen, J. (2005). *Graph Theory and Its Applications*. Textbooks in Mathematics (2nd ed.). Chapman & Hall/CRC.

[29] Groves, R. M., Fowler Jr., F. J., Couper, M. P., Lepkowski, J. M., Eleanor, S., & Tourangeau, R. (2009). *Survey Methodology*. (2nd ed.). Wiley.

[30] Halmos, P. (1960). *Naive Set Theory*. Van Nostrand.

[31] Kim, H. J., Cox, L. H., Karr, A. F., Reiter, J. P., & Wang, Q. (2015). Simultaneous edit-imputation for continuous microdata. *Journal of the American Statistical Association*, *110*, 987–999.

[32] Liepins, G. E. (1980). *Refinements to the Boolean Approach to Automatic Data Editing*. Technical Report Oak Ridge National Laboratory.

[33] Liepins, G. E. (1981). *A Rigorous, Systematic Approach to Automatic Data Editing and its Statistical Basis*. Technical Report Oak Ridge National Laboratory.

[34] Liepins, G. E. (1984). *Algorithms for Error Localization of Discrete Data*. Technical Report Oak Ridge National Laboratory.

[35] Lincoff, G. (1981). *National Audubon Society Field Guide to North American Mushrooms*. Knopf.

[36] Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2012). Isolation-based anomaly detection. *ACM Trans. on Knowledge Discovery from Data*, *6*.

[37] Manrique-Vallier, D., & Reiter, J. P. (2017). Bayesian simultaneous edit and imputation for multivariate categorical data. *Journal of the American Statistical Association*, *112*, 1708–1719.

[38] Nordbotten, S. (1955). Measuring the error of editing the questionnaires in a census. *Journal of the American Statistical Association*, *50*, 364–369.

[39] Nordbotten, S. (1963). Automatic editing of individual statistical observations. In *Statistical Standards and Studies* 2.

[40] Qahtan, A., Tang, N., Ouzzani, M., Cao, Y., & Stonebraker, M. (2020). Pattern functional dependencies for data cleaning. *Proc. of the VLDB Endowment*, *13*, 684–697.

[41] Rammelaere, J., Geerts, F., & Goethals, B. (2017). Cleaning data with forbidden itemsets. In *Proc. of the 33rd IEEE International Conference on Data Engineering (ICDE 2017)* (pp. 897–908). IEEE.

[42] Redman, T. C. (1997). *Data Quality for the Information Age*. (1st ed.). Norwood, MA, USA: Artech House, Inc.

[43] Rekatsinas, T., Chu, X., Ilyas, I. F., & Ré, C. (2017). Holoclean: Holistic data repairs with probabilistic inference. *Proc. of the VLDB Endowment*, *10*, 1190–1201.

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

[45] Sunaga, T. (1958). Theory of an interval algebra and its application to numerical analysis. *RAAG Memoirs*, *2*, 29–46.

[46] Winkler, W. E. (1995). Editing discrete data. In *Proc. of the Survey Research Methods Section* (pp. 108–113). American Statistical Association.

[47] Winkler, W. E. (1997). Set covering and editing discrete data. In *Proc. of the Survey Research Methods Section* (pp. 564–569). American Statistical Association.

[48] Winkler, W. E. (1999). *State of Statistical Data Editing and Current Research Problems*. Technical Report.