Cleaning Data with Forbidden Itemsets

Joeri Rammelaere
University of Antwerp
Middelheimlaan 1
Antwerp, Belgium
joeri.rammelaere@uantwerp.be

Floris Geerts
University of Antwerp
Middelheimlaan 1
Antwerp, Belgium
floris.geerts@uantwerp.be

Bart Goethals
University of Antwerp
Middelheimlaan 1
Antwerp, Belgium
bart.goethals@uantwerp.be

***Abstract*—Methods for cleaning dirty data typically rely on additional information about the data, such as user-specified constraints that specify when a database is dirty. These constraints often involve domain restrictions and illegal value combinations. Traditionally, a database is considered clean if all constraints are satisfied. However, many real-world scenario's only have a dirty database available. In such a context, we adopt a dynamic notion of data quality, in which the data is clean if an error discovery algorithm does not find any errors. We introduce *forbidden itemsets* which capture unlikely value co-occurrences in dirty data, and we derive properties of the lift measure to provide an efficient algorithm for mining low lift forbidden itemsets. We further introduce a repair method which guarantees that the repaired database does not contain any low lift forbidden itemsets. The algorithm uses nearest neighbor imputation to suggest possible repairs. Optional user interaction can easily be integrated into the proposed cleaning method. Evaluation on real-world data shows that errors are typically discovered with high precision, while the suggested repairs are of good quality and do not introduce new forbidden itemsets, as desired.**

I. Introduction

In recent years, research on detecting inconsistencies in data has focused on a constraint-based data quality approach: a set of constraints in some logical formalism is associated with a database, and the data is considered consistent or clean if and only if all constraints are satisfied. Many such formalisms exist, capturing a wide variety of inconsistencies, and systematic ways of repairing the detected inconsistencies are in place. A frequently asked question is, "where do these constraints come from?". The common answer is that they are either supplied by experts, or automatically discovered from the data [1], [2]. In most real-world scenario's, however, the underlying data is dirty. This raises concerns about the reliability of the discovered constraints.

Assume for the moment that the constraints are reliable and are used to repair (clean) a dirty database. For the sake of the argument, what if one re-runs the constraint discovery algorithm on the repair and finds other constraints? Does this imply that the repair is not clean after all, or that the discovery algorithm may in fact find unreliable constraints? It is a typical chicken or the egg dilemma. The problem is that constraints are considered to be *static*: once found, they are treated as a gold standard. To remedy this, we propose a *dynamic* notion of data quality. The idea is simple:

> "We consider a given database to be clean if a constraint discovery algorithm does not detect any violated constraints on that data."

Constraints thus reflect inconsistencies which depend on the actual data. Clearly, this dynamic notion presents a new challenge when repairing, since the constraints may shift during repairs. Indeed, it does not suffice to only resolve inconsistencies for constraints found on the original dirty data, one also has to ensure that no new constraints (and thus new inconsistencies) are found on the repaired data. Note that we focus on repairing data under dynamic constraints, as opposed to cleaning dynamic data. To our knowledge, this is a new view on data quality, raising many interesting challenges.

The main contribution of this paper is to illustrate this dynamic view on data quality for a particular class of constraints. In particular, we consider errors that can be caught by so-called *edits*, which is "the" constraint formalism used by census bureaus worldwide [3], [4] and can be seen as a simple class of denial constraints [5], [6]. Intuitively, an edit specifies *forbidden value combinations*. For example, an age attribute cannot take a value higher than 130, a city can only have certain zip codes, and people of young age cannot have a driver's license. These edits are typically designed by domain experts and are generally accepted to be a good constraint formalism for detecting errors that occur in single records. However, we aim to automatically discover such edits in an unsupervised way by using pattern mining techniques. To make the link to pattern mining more explicit and to differentiate from edits, we call our patterns *forbidden itemsets*. Although our technique is designed such that human supervision is not compulsory, we show that optional user interaction can be readily integrated to improve the reliability of the methods.

Pattern mining techniques are typically used to uncover positive associations between items, measured by different interestingness measures such as frequency, confidence, lift, and many others. Based on experience, *discovered patterns often reveal errors in the data* in addition to interesting associations. For example, an association rule which holds for 99% of the data could be interesting in itself, but might also represent a well-known dependency in the data which should hold for 100%. The fact that the association doesn't hold for 1% of the data is then more interesting. This 1% of the data often points to unlikely co-occurrences of values in the data, which forbidden itemsets aim to capture. In order to reliably detect unlikely co-occurrences, it is clear that a large body of clean data is needed. We therefore focus on low error rate data, such as census data or data that underwent some curation [7].

Apart from detecting errors we also aim to provide suggestions for how to repair them, i.e., suggest modifications to the data such that after these modifications, no new forbidden

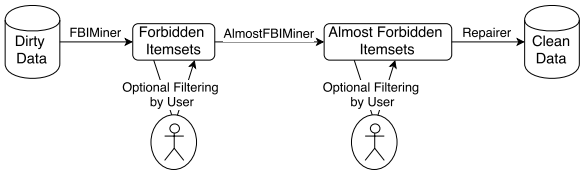

Figure 1. Schematic overview of the proposed cleaning mechanism.

itemsets are found. Here we again take inspiration from census data imputation methods that assume the presence of enough clean data [4] and take suggested modifications from similar, clean objects. The availability of clean data is commonly assumed in repairing methods, either as a large part of the input data or, for example, in the form of master data [8], [9].

Figure 1 gives a schematic overview of the proposed cleaning mechanism in its entirety. We capture unlikely co-occurrences by means of forbidden itemsets. Our algorithm FBIMINER employs pruning strategies to optimize the discovery of forbidden itemsets. Linking back to the beginning of the introduction, we will regard data to be dirty if FBIMINER finds forbidden itemsets in the data. Users may optionally filter out forbidden itemsets by declaring them as valid. Furthermore, we also devise a *repairing algorithm* that repairs a dirty dataset and ensures that no forbidden itemsets exist in the repair, hence it is indeed clean. To achieve this, we consider so-called *almost forbidden itemsets* and present an algorithm A-FBIMINER for mining them. Again, users can optionally filter out such itemsets. All algorithms are experimentally validated.

Organization of the paper. The paper is organized as follows: In Sect. II we discuss the most relevant related work. Notations and basic concepts are presented in Sect. III, followed by a formal problem statement in Sect. IV. Section V introduces forbidden itemsets, their properties, and the FBIMINER algorithm. Section VI presents the repair algorithm, focusing on our strategy to avoid new inconsistencies. The possibility of user interaction is discussed in Sect. VII. In Sect. VIII we show experimental results, before we pose a conclusion in Sect. IX. Proofs and additional plots can be found in the appendix of the full version of the paper [10].

II. Related Work

There has been a substantial amount of work on constraint-based data quality in the database community (see [1], [2] for recent surveys). Most relevant to our work are constraints that concern single tuples such as constant conditional functional dependencies [1] and constant denial constraints [5], [6]. Algorithms are in place to (i) discover these constraints from data [11], [12], [1], [13]; and (ii) repair the errors, once the constraints are fixed [14], [15], [16], [5]. Moreover, user interaction is often used to guide the repairing algorithms [17], [18], [19] and statistical methods are in place to measure the impact of repairing [20]. As previously mentioned, all these methods use a static notion of cleanliness. Our notion of forbidden itemsets is closest in spirit to edits, used by census bureaus [3] and our repairing method is similar to hot-deck imputation methods [4]. Capturing and detecting inconsistencies is also closely related to anomaly and outlier detection (see [21], [22], [23] for recent surveys). A recent study [24] provides a comparison of detection methods. In the pattern mining community many different interestingness measures exist. We mention [25] in which outliers are discovered using a measure that is similar to ours. Furthermore, Error-Tolerant Itemsets [26], [27] can be regarded as the inverse of the forbidden itemsets. Compared to these methods, we identify new properties of the lift measure to speed up the discovery, and use forbidden itemsets to both detect *and repair* data.

III. Preliminaries

We consider datasets $\mathcal{D}$ consisting of a finite collection of objects. An *object* o is a pair $\langle \mathsf{oid}, I \rangle$ where oid is an *object identifier*, e.g., a natural number, and I is an itemset. An *itemset* is a set of *items* of the form (A, v), where A comes from a set $\mathcal{A}$ of *attributes* and v is a *value* from a finite domain of categorical values $\mathsf{dom}(A)$ of A. An itemset contains at most one item for each attribute in $\mathcal{A}$. We define the value of an object $o = \langle \mathsf{oid}, I \rangle$ in attribute A, denoted by $o[A]$, as the value v where $(A, v) \in I$, and let $o[A]$ be undefined otherwise. We denote by $\mathcal{I}$ the set of all attribute/value pairs (A, v).

An object $o = \langle \mathsf{oid}, I \rangle$ is said to *support* an itemset J if $J \subseteq I$, i.e., J is contained in I. The *cover* of an itemset J in $\mathcal{D}$, denoted by $\mathsf{cov}(J, \mathcal{D})$, is the set of oid's of objects in $\mathcal{D}$ that support J. The *support* of J in $\mathcal{D}$, denoted by $\mathsf{supp}(J, \mathcal{D})$, is the number of oid's in its cover in $\mathcal{D}$. Similarly, the *frequency* of an itemset J in $\mathcal{D}$ is the fraction of oid's in its cover: $\mathsf{freq}(J, \mathcal{D}) = \mathsf{supp}(J, \mathcal{D})|/|\mathcal{D}|$, where $|\mathcal{D}|$ is the number of objects in $\mathcal{D}$. We sometimes represent $\mathcal{D}$ in a *vertical data layout* denoted by $\mathcal{D}\downarrow$. Formally, $\mathcal{D}\downarrow = \{(i, \mathsf{cov}(\{i\}, \mathcal{D})) \mid i \in \mathcal{I}, \mathsf{cov}(\{i\}, \mathcal{D}) \neq \emptyset\}$. Clearly, one can freely go from $\mathcal{D}$ to $\mathcal{D}\downarrow$, and vice versa.

We assume that a similarity measure between objects is given, and denote by $\mathsf{sim}(o, o')$ the similarity between objects o and o'. The similarity between two datasets $\mathcal{D}$ and $\mathcal{D}'$ is denoted by $\mathsf{sim}(\mathcal{D}, \mathcal{D}')$. Any similarity measure can be used in our setting. We describe the similarity function used in our experiments in Sect. VIII.

IV. Problem Statement

We first phrase our problem in full generality (following [28]) before making things more specific in the next section. Consider a dataset $\mathcal{D}$ and some *constraint language* $\mathcal{L}$ for expressing properties that indicate *dirtiness* in the data, e.g., $\mathcal{L}$ could consist of conditional functional dependencies [1], edits [4], or association rules [29]. Furthermore, let q be a *selection predicate* (evaluating to true or false) that assesses the relevance of constraints $\varphi \in \mathcal{L}$ in $\mathcal{D}$. For example, when φ is a conditional functional dependency, $q(\mathcal{D}, \varphi)$ may return true if φ is violated in $\mathcal{D}$. We denote by $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ the set of all *dirty constraints*, i.e., all $\varphi \in \mathcal{L}$ for which $q(\mathcal{D}, \varphi)$ evaluates to true. For example, $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ may consist of all violated conditional functional dependencies, all edits that apply to an object, or all low confidence association rules.

Definition 1. A dataset $\mathcal{D}$ is said to be *clean* relative to language $\mathcal{L}$ and predicate q if $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ is empty; $\mathcal{D}$ is called *dirty* otherwise. □

With this definition we take a *completely new view* on data quality. Indeed, existing work in this area [1] typically *fixes* the constraints up front, regardless of the data. For example,

Table I. EXAMPLE FORBIDDEN ITEMSETS FOUND IN UCI DATASETS

Forbidden Itemsets	Dataset	τ
Sex=Female, Relationship=Husband Sex=Male, Relationship=Wife Age=<18, Marital-status=Married-c-s Age=<18, Relationship=Husband Relationship=Not-in-family, Marital=Married-c-s	Adult	0.01
aquatic=0, breathes=0 (clam) type=mammal, eggs=1 (platypus) milk=1, eggs=1 (platypus) type=mammal, toothed=0 (platypus) eggs=0, toothed=0 (scorpion) milk=1, toothed=0 (platypus) tail=1, backbone=0 (scorpion)	Zoo	0.1
bruises=t, habitat=l population=y, cap-shape=k cap-surface=s, odor=n, habitat=d cap-surface=s, gill-size=b, habitat=d edible=e, habitat=d, cap-shape=k	Mushroom	0.025

edits are often designed by experts and then compared with the data. In our definition, we only specify the *class* of constraints, e.g., edits. Which edits are used for declaring the data clean or dirty *depends entirely on the underlying data*. We thus introduce a *dynamic* rather than a static notion of dirtiness/cleanliness of data: when the data changes, so do the edits under consideration. With this notion at hand, we are next interested in repairs of the data. Intuitively, a repair of a dirty dataset is a modified dataset that is clean.

Definition 2. Given datasets $\mathcal{D}$ and $\mathcal{D}'$, language $\mathcal{L}$, predicate q and similarity function sim, we say that $\mathcal{D}'$ is an $(\mathcal{L}, q)$-*repair* if (i) $\mathcal{D}'$ has the same set of object identifiers as $\mathcal{D}$; and (ii) $\mathsf{dirty}(\mathcal{D}', \mathcal{L}, q)$ is empty. An $(\mathcal{L}, q)$-repair $\mathcal{D}'$ is *minimal* if $\mathsf{sim}(\mathcal{D}, \mathcal{D}')$ is maximal amongst all $(\mathcal{L}, q)$-repairs of $\mathcal{D}$. □

V. FORBIDDEN ITEMSETS

We first specialize constraint language $\mathcal{L}$ and predicate q such that $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ corresponds to *inconsistencies* in $\mathcal{D}$. We define $\mathcal{L}$ as the class of itemsets and define q such that $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ corresponds to what we will call *forbidden itemsets* (Sect. V-A). Intuitively, these are itemsets which do occur in the data, despite being very improbable with respect to the rest of the data. Furthermore, we show how to compute $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ for *low lift* forbidden itemsets. As such itemsets are typically *infrequent*, existing itemset mining algorithms are not optimized for this task. We therefore derive properties of the lift measure that allow for substantial pruning when mining forbidden itemsets (Sect. V-B). We conclude by presenting a version of the well-known Eclat algorithm [30], enhanced with our derived pruning strategies and optimizations specific for the task of mining low lift forbidden itemsets (Sect. V-C).

Before formally introducing forbidden itemsets as a constraint language $\mathcal{L}$, let us provide some additional motivation for considering invalid or unlikely value combinations (as represented by forbidden itemsets) as error detection formalism. First of all, invalid value combinations have been used for decades to detect errors in census data starting with the seminal work by Fellegi and Holt [3]. Second, although more expressive formalisms such as conditional functional dependencies (CFDs) [31] and denial constraints (DCs) [5], [6] have become popular for error detection and repairing, many constraints used in practice are very simple. As an example, most of the constraints reported in Table 4 in [24] can be regarded as constraints that only involve constants. This is clear for "checks" that specify invalid domain values. However, even a functional dependency such as $(\mathit{zip} \rightarrow \mathit{state})$ can be regarded as a (finite) collection of constant rules that associate specific zip codes to state names. The violations of these rules clearly are invalid value combinations. Similarly, almost half of the DCs reported in [6] only involve constants and concern single tuples. It therefore seems natural to first gain a better understanding of these simple constraints in our dynamic data quality setting. Finally, the discovery of CFDs and DCs [1], [11], [12] in their full generality is very slow due to the high expressiveness of these constraint languages. Experiments show that the discovery may take up to hours on a single machine. This makes general constraints infeasible in settings such as ours, where interactivity or quick response times are needed. For all these reasons, we believe that forbidden itemsets provide a good balance between expressiveness, efficiency of discovery, and efficacy in error detection. Furthermore, they are easily interpretable, allowing users to inspect and filter out false positives, as discussed in Sect. VII.

A. Low Lift Itemsets

We consider $\mathcal{L}$ consisting of the class of itemsets and want to define q such that for an itemset I, $q(\mathcal{D}, I)$ is true if I corresponds to a possible inconsistency in the data. In general, we can use a *likeliness function* $L : 2^{\mathcal{I}} \times \mathcal{D} \rightarrow \mathbb{R}$ that indicates how likely the occurrence of an itemset is in $\mathcal{D}$. Such a likeliness function can be defined to accommodate various types of constraints on value combinations within a single tuple. If we denote by τ a maximum likeliness threshold, then we define *τ-forbidden itemsets* as follows.

Definition 3. Let $\mathcal{D}$ be a dataset, L a likeliness function, I an itemset and τ a maximum likeliness threshold. Then, I is called a *τ-forbidden itemset* whenever $L(I, \mathcal{D}) \leq \tau$. □

Phrased in the general framework from the previous section, we thus have that $\mathcal{L}$ is the class of itemsets and $q(\mathcal{D}, I)$ is true if I is a τ-forbidden itemset. Hence, $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ consists of all τ-forbidden itemsets in $\mathcal{D}$.

In this paper, we propose to use the *lift* measure of an itemset as likeliness function. Intuitively, it gives an indication of how probable the co-occurrence of a set of items is given their separate frequencies. Lift is generally used as an interestingness measure in association rule mining, where rules with a high lift between antecedent and consequent are considered the most interesting [25], and has also been used for constraint discovery [11]. A straightforward extension of lift from rules to itemsets assumes "full" independence among all individual items [32, p. 310]. In other words, an itemset I is regarded as "unlikely" when $\mathsf{freq}(I, \mathcal{D})$ is much smaller than $\mathsf{freq}(\{i_1\}, \mathcal{D}) \times \cdots \times \mathsf{freq}(\{i_k\}, \mathcal{D})$, where $i_1, \ldots, i_k$ are the items in I. However, this introduces an undesirable bias towards larger itemsets: many items with a slight negative correlation might have a lower lift than two items with a strong negative correlation.

Instead of full independence, we adopt "pairwise" independence, in which $\mathsf{freq}(I, \mathcal{D})$ is compared against $\mathsf{freq}(J, \mathcal{D}) \times \mathsf{freq}(I \setminus J, \mathcal{D})$ for any non-empty $J \subset I$, and the *maximal* discrepancy ratio is taken as the *lift* of I in $\mathcal{D}$:

Definition 4. Let $\mathcal{D}$ be a dataset and let I be an itemset. The *lift* of I, denoted by $\mathsf{lift}(I, \mathcal{D})$, is defined as

$$\mathsf{lift}(I, \mathcal{D}) := \frac{|\mathcal{D}| \times \mathsf{supp}(I, \mathcal{D})}{\min\limits_{\emptyset \subset J \subset I} \big\{\mathsf{supp}(J, \mathcal{D}) \times \mathsf{supp}(I \setminus J, \mathcal{D})\big\}} \quad \square$$

We note that this definition is conceptually related to association rules and has been used successfully in the context of redundancy [25] and outlier detection [33], two concepts very similar in spirit to our intended notion of inconsistencies.

One could further generalize the notion of lift by ranging over partitions of I consisting of more than two parts, full independence being a special case in which I is partitioned in all its items. We find, however, that pairwise independence is already effective for detecting unlikely value combinations and is more efficient to compute.

From now on, we refer to *τ-forbidden itemsets* as itemsets I for which $\mathsf{lift}(I, \mathcal{D}) \leqslant \tau$ holds, following Def. 3. When using lift, τ will typically be small, and we assume that $\tau < 1$.

Example 1. To illustrate that τ-forbidden itemsets are an effective formalism for capturing unlikely value combinations, we show some example forbidden itemsets found in UCI datasets in Table I. In the Adult dataset, the co-occurrence of Female and Husband, as well as Male and Wife, are clearly erroneous. Other examples involve a married person under the age of 18 and people who are married, yet living in with an unrelated household. In the Zoo dataset, the first forbidden itemset shows that the animal *clam* in the dataset is neither aquatic nor breathing. To our knowledge, clams are in fact aquatic, so these values are indeed in error. The other forbidden itemsets detect animals that are in some way an exception in nature. For example, the *platypus* is famous for being one of few existing mammal species that lays eggs, the other species being anteaters. Similar exceptional combinations are encountered in the other UCI datasets, such as the Mushroom dataset, although the forbidden itemsets are more difficult to interpret for this dataset. While not all of these examples represent actual *errors*, they show that the forbidden itemsets are capable of detecting peculiar objects that require extra attention. Of course, it makes little sense to repair objects such as the platypus. Typically, user validation of the discovered errors will be preferable over fully automatic repairs. This is addressed in Sect. VII. $\diamondsuit$

B. Properties of the Lift Measure

Before presenting our algorithm FBIMINER that mines τ-forbidden itemsets, we describe some properties of the lift measure that underlie our algorithm.

While the lift measure is neither monotonic nor anti-monotonic, two properties that are typically used for pruning in pattern mining algorithms, it still has some properties that allow the pruning of large branches of the search tree: since a low lift requires that an itemset occurs *much less often* than its subsets, we can use the relation between the support of a τ-forbidden itemset and the support of its subsets to prune. As we will explain shortly, FBIMINER performs a depth-first search for forbidden itemsets. To make this search efficient, pruning strategies should be in place that discard all supersets of a particular itemset. For this purpose, we derive properties that must hold for all subsets of a τ-forbidden itemset.

Given an itemset I we denote by σ_I^{max} the highest support of an item $\{i\}$ in I or any of I's supersets J, i.e, σ_I^{max} is the highest support in I's branch of the search tree. More formally,

$$\sigma_I^{max} := \max\{\mathsf{supp}(\{i\}, \mathcal{D}) \mid i \in J, I \subseteq J\}.$$

This quantity can be used to obtain a lower bound on the support of subsets of J when J is a τ-forbidden itemset:

Proposition 1. *For any two itemsets I and J such that $I \subset J$, if J is a τ-forbidden itemset then* $\mathsf{supp}(I, \mathcal{D}) \geq \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D})}{\sigma_I^{max} \times \tau}$. $\square$

Furthermore, for any τ-forbidden itemset J in the dataset, it trivially holds that $\mathsf{supp}(J, \mathcal{D}) \geq 1$ and thus any itemset $I \subset J$ must have $\mathsf{supp}(I, \mathcal{D}) \geq \frac{|\mathcal{D}|}{\sigma_I^{max} \times \tau}$. This implies that in the depth-first search, it suffices to expand itemsets I for which $\mathsf{supp}(I, \mathcal{D}) \geq \frac{|\mathcal{D}|}{\sigma_I^{max} \times \tau}$.

Furthermore, Prop. 1 can be leveraged to show that a minimum reduction in support between subsets of a τ-forbidden itemset is required.

Proposition 2. *For any three itemsets I, J and K such that $I \subset J \subseteq K$ holds, if K is a τ-forbidden itemset, then* $\mathsf{supp}(I, \mathcal{D}) - \mathsf{supp}(J, \mathcal{D}) \geq \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|} > 0$. $\square$

In particular, for J to be τ-forbidden, we must have that $\mathsf{supp}(I, \mathcal{D}) - \mathsf{supp}(J, \mathcal{D}) \geq \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|} > 0$ holds for any of its subsets I. During the depth-first search, when expanding I to J, if this condition is not satisfied, then J and all of its supersets K can be pruned. Furthermore, the proposition implies that τ-forbidden itemsets are so-called generators [34], i.e., if J is τ-forbidden then $\mathsf{supp}(I, \mathcal{D}) > \mathsf{supp}(J, \mathcal{D})$ for any $I \subset J$. A known property of generators is that all their subsets are generators as well, meaning that the entire subtree can be pruned if a non-generator is encountered during the search.

Our next pruning method uses the lift of an itemset to bound the support of its supersets. Indeed, the denominator of the lift measure is in fact anti-monotonic.

Proposition 3. *For any two itemsets I and J such that $I \subset J$, it holds that:*

$$\mathsf{lift}(J, \mathcal{D}) \geq \frac{\mathsf{supp}(J, \mathcal{D}) \times |\mathcal{D}|}{\min\limits_{S \subset I} \big\{\mathsf{supp}(S, \mathcal{D}) \times \mathsf{supp}(I \setminus S, \mathcal{D})\big\}}. \quad \square$$

Clearly, this lower bound can be used to prune itemsets J that cannot lead to τ-forbidden itemsets. Note that to use the lower bound, one needs a lower bound on $\mathsf{supp}(J, \mathcal{D})$. We again use the trivial lower bound $\mathsf{supp}(J, \mathcal{D}) \geq 1$.

Finally, since itemsets with low lift are obtained when they occur much less often than their subsets, it is expected that such forbidden itemsets will have a low support. In fact, one can precisely characterize the maximal frequency of a τ-forbidden itemset.

Proposition 4. *If I is a τ-forbidden itemset then its frequency is bounded by* $\mathsf{freq}(I, \mathcal{D}) \leqslant \frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$. *Furthermore, for small τ-values this bound converges (from above) to $\frac{\tau}{4}$.* $\square$

Algorithm 1 An Eclat-based algorithm for mining low lift τ-forbidden itemsets

```
1:  procedure FBIMINER(D↓, I ⊆ 𝓘, τ)
2:    FBI ← ∅
3:    for all i ∈ 𝓘 occurring in 𝒟 in reverse order do
4:      J ← I ∪ {i}
5:      if not isGenerator(J) then
6:        continue
7:      storeGenerator(J)
8:      if |J| > 1 & freq(J, 𝒟) ≤ 2/τ − 2√(1/τ² − 1/τ) − 1 then
9:        if a subset of J has been pruned then
10:         continue
11:       if lift(J, 𝒟) ⩽ τ then
12:         FBI ← FBI ∪ {J}
13:       if |𝒟|/τ > min_{S⊂J} {supp(S, 𝒟) × supp(J \ S, 𝒟)} then
14:         continue
15:     if supp(J, 𝒟) < |𝒟| / (σ_J^max × τ) then
16:       continue
17:     𝒟↓[i] ← ∅
18:     for all j ∈ 𝓘 in 𝒟 such that j > i do
19:       C ← cov({i}, 𝒟) ∩ cov({j}, 𝒟)
20:       if supp(J, 𝒟) − |C| ≥ (1/τ) − (σ_J^max/|𝒟|) then
21:         if |C| > 0 then
22:           𝒟↓[i] ← 𝒟↓[i] ∪ {(j, C)}
23:     FBI ← FBI ∪ FBIMINER(𝒟↓[i], J, τ)
24:   return FBI
```

The proposition tells that for small values of τ, τ-forbidden itemsets are at most approximately $\tau/4$-frequent and that itemsets whose frequency exceeds $\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ cannot be τ-forbidden. This result can be used to obtain an initial estimate for τ: Clearly, this bound should be at least 1, yet not too high, such that frequent itemsets cannot be forbidden.

C. Forbidden Itemset Mining Algorithm

We now present an algorithm, FBIMINER, for mining τ-forbidden itemsets in a dataset $\mathcal{D}$. That is, the algorithm computes $\mathsf{dirty}(\mathcal{D}, \mathcal{L}, q)$ for the language and predicate defined earlier. The algorithm is based on the well-known Eclat algorithm for frequent itemset mining [30]. Here, we only describe the main differences with Eclat. The pseudo-code of FBIMINER is shown in Alg. 1. The algorithm is initially called with $\mathcal{D}\downarrow$ ($\mathcal{D}$ in vertical data layout), $I = \emptyset$ and the lift threshold τ. Just like Eclat, FBIMINER employs a depth-first search of the itemset lattice (`for` loop line 3 – 23, and recursive call on line 23) using set intersections of the covers of the items to compute the support of an itemset (line 19). When expanding an itemset I in the search tree (line 4), new itemsets are generated by extending I with all items in the dataset that occur in the objects in I's cover (lines 18 – 22). Furthermore, these items are added according to some total ordering on the items, i.e., items are only added when they come after each item in I (line 18). Items are ordered by ascending support, as this is known to improve efficiency.

A first challenge is to tweak the Eclat algorithm such that the lift of itemsets can be computed. Observe that the lift of an itemset is dependent on the support of *all* of its subsets. For this purpose, we use the same depth-first traversal as Eclat, but traverse it in reverse order (line 3). Indeed, such a reverse pre-order traversal of the search space visits all subsets of an itemset J before visiting J itself [35]. This is exactly what is required to compute the lift measure, provided that the support of each processed itemset is stored. However, Eclat generates a candidate itemset based on only two of its subsets [30]. Hence, the supports of all subsets of an itemset are not immediately available in the search tree. To remedy this, we store the support of the processed itemsets using a prefix tree, for time- and memory-efficient lookup during lift computation.

Having integrated lift computation in the algorithm, we next turn to our pruning and optimization strategies. We deploy four pruning strategies (lines 9, 13, 15 and 20). The first strategy (line 9) applies to itemsets J for which the lift cannot be computed. This happens when some of its subsets are pruned away in an earlier step. Since itemsets are only pruned when *none* of their supersets can be τ-forbidden, this implies that J cannot be τ-forbidden. The absence of subsets is detected when the lift computation requests the support of a subset that is not stored. Our pruning then implies that J and its supersets cannot be τ-forbidden and thus can be pruned (line 7).

The second pruning strategy (line 13) applies to itemsets J for which we have been able to compute their lift. Indeed, when $\frac{|\mathcal{D}|}{\tau} > \min_{S \subset J} \{\mathsf{supp}(S, \mathcal{D}) \times \mathsf{supp}(J \setminus S, \mathcal{D})\}$ then Prop. 3 tells that J cannot be a subset of a τ-forbidden itemset. Hence, all itemsets in the tree below J are pruned (line 14).

By contrast, the third strategy (line 15) leverages Prop. 1 and skips supersets of itemsets J, regardless of whether its lift was computed. Indeed, when $\mathsf{supp}(J, \mathcal{D}) < \frac{|\mathcal{D}|}{\sigma_J^{max} \times \tau}$ holds then J cannot be part of a τ-forbidden itemset, resulting in a further pruning of the search space.

The fourth strategy employs Prop. 2 to prune extensions of J that do not cause a sufficient reduction in support. This check is performed prior to the recursive call (line 20).

Finally, we also implement an optimization that avoids certain lift computations (line 8). Only when the algorithm encounters an itemset J with at least two items and a frequency lower than the bound from Prop. 4, the lift of J is computed. All other itemsets cannot be τ-forbidden by Prop. 4. Note that this only eliminates the need for checking the lift of certain itemsets but by itself does not lead to a pruning of its supersets.

A careful reader may have spotted the optimized pruning of non-generators on lines 5-7. Recall that as a direct consequence of Prop. 2, any τ-forbidden itemset must be a generator, i.e., have a support which is strictly lower than that of all of its subsets. The Talky-G algorithm [36] implements specific optimizations for mining such generators, using a hash-based method that was introduced in the Charm algorithm for closed itemset mining [37]. We use the same technique in FBIMINER to efficiently prune non-generators.

During the mining process, all encountered generators are stored in a hashmap (procedure `storeGenerator` on line 7). As hash value we use, just like the Charm algorithm, the sum of the oid's of all objects in which an itemset occurs. If an itemset has the same support as one of its subsets, it is clear that this itemset must occur in exactly the same objects, and

will map to the same hash value. Moreover, the probability of unrelated itemsets having the same *sum of oids* is lower than the probability of them having the same support. Therefore this sum is taken as hash value instead of the support of itemsets.

Procedure `isGenerator` on line 5 checks all stored itemsets with the same hash value as J. If any of these itemsets are a subset of J with the same support, J is discarded as a non-generator. Furthermore, since all supersets of a non-generator are also non-generators, the entire subtree can be pruned. If no subset with identical support is discovered for an itemset J, then J is either a generator, or a subset with identical support has previously been pruned, in which case J will eventually be pruned on line 9.

VI. Repairing Inconsistencies

The algorithm FBIMiner discovers a set of τ-forbidden itemsets that describe inconsistencies in $\mathcal{D}$. When this set is non-empty, $\mathcal{D}$ is regarded as dirty. We next want to clean $\mathcal{D}$. Following the general framework outlined in section IV, we wish to compute a repair $\mathcal{D}'$ of $\mathcal{D}$ such that (i) $\mathcal{D}'$ is clean, i.e., no τ-forbidden itemsets should be found in $\mathcal{D}'$; and (ii) $\mathcal{D}'$ differs minimally from $\mathcal{D}$. Due to the dynamic notion of data quality, (i) becomes more challenging than in a traditional repair setting. We choose not to optimize (ii) directly by computing minimal modifications to dirty objects, but instead impute values from clean objects that differ minimally from a dirty object, in line with common practice in data imputation. We start by showing how the creation of new τ-forbidden itemsets can be avoided by means of so-called almost forbidden itemsets (Sect. VI-A) and explain how these can be mined (Sect. VI-B), before describing the repair algorithm itself (Sect. VI-C).

A. Ensuring a Clean Repair

Given a dataset $\mathcal{D}$ and its set of τ-forbidden itemsets $\mathsf{FBI}(\mathcal{D},\tau)$, we define the *dirty* objects in $\mathcal{D}$, denoted by $\mathcal{D}_{dirty}$, as those objects in $\mathcal{D}$ that appear in the cover of an itemset in $\mathsf{FBI}(\mathcal{D},\tau)$. In other words, $\mathcal{D}_{dirty}$ consists of all objects that support a τ-forbidden itemset. The remaining set of clean objects in $\mathcal{D}$ is denoted by $\mathcal{D}_{clean}$. The repair algorithm will produce a dataset $\mathcal{D}'$ by modifying all objects in $\mathcal{D}_{dirty}$ to remove the forbidden itemsets. We consider value modifications where values come from clean objects. Recall however that we need to obtain a repair $\mathcal{D}'$ of $\mathcal{D}$ such that $\mathsf{FBI}(\mathcal{D}',\tau)$ is empty, i.e., such that $\mathcal{D}'$ is clean. We first present an example to show that this is not a trivial problem.

Example 2. People typically graduate High School in the year they turn 18. Depending on the timing of a census, there may be graduates who are still only 17 years old. In the Adult Census dataset, the itemset (Age=<18, Education=HS-grad) is rare, with a lift ≈ 0.072. Assume that τ-forbidden itemsets were mined with $\tau = 0.07$. The itemset is thus not considered forbidden, and rightly so. However, an object containing (Age=<18, Education=HS-grad) could, for example, contain the forbidden itemset (Age=<18, MaritalStatus=Divorced), where MaritalStatus is in error. If the repair algorithm were to change *Age* instead, the lift of (Age=<18, Education=HS-grad) would drop to ≈ 0.068! This itemset will then become τ-forbidden in $\mathcal{D}'$, yielding again a dirty dataset. $\diamondsuit$

A naive approach for avoiding new inconsistencies would be to run FBIMiner on $\mathcal{D}'$ for each candidate modification, and reject the modification in case $\mathsf{FBI}(\mathcal{D}',\tau)$ is non-empty. In view of the possibly exponential number of candidate modifications, this approach is not feasible for all but the smallest datasets. As an alternative, we propose to compute up front enough information to ensure cleanliness of multiple repairs. More specifically, the procedure A-FBIMiner computes a set $\mathbb{A}$ of *almost τ-forbidden itemsets*, i.e., itemsets that could become τ-forbidden after a given number of modifications k. This computation relies only on the dataset $\mathcal{D}$ and its dirty objects, and not on the particular modifications made during repairing.

▷ **Almost Forbidden Itemsets.** Almost forbidden itemsets are mined by algorithm A-FBIMiner. It mines itemsets J similarly to FBIMiner, but uses a relaxed notion of lift, called the *minimal possible lift of J after k modifications*, to be explained below. More specifically, by using the minimal possible lift measure, A-FBIMiner returns a set $\mathbb{A}$ of itemsets such that for *any* dataset $\mathcal{D}'$ obtained from $\mathcal{D}$ by at most k modifications,

$$\mathsf{FBI}(\mathcal{D}',\tau) \subseteq \mathbb{A}. \qquad (\dagger)$$

We next explain what precisely $\mathbb{A}$ consists of and how it can be used to avoid new inconsistencies whilst repairing. It is crucial for our approach that $\mathbb{A}$ accommodates for *any* repair $\mathcal{D}'$ of $\mathcal{D}$ obtained from at most k modifications as it eliminates the need for considering all possible repairs one by one.

▷ **Minimal Possible Lift.** To define the relaxed lift measure used by A-FBIMiner, we consider the following problem: *Given an itemset J in $\mathcal{D}$ and its $\mathsf{lift}(J,\mathcal{D})$, can J become τ-forbidden after k modifications to $\mathcal{D}$ have been made*? Let us first analyze the minimal possible lift in case a single modification is made. Suppose that $\mathsf{lift}(J,\mathcal{D}) = |\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})/(\mathsf{supp}(I,\mathcal{D}) \times \mathsf{supp}(J \setminus I,\mathcal{D}))$ for some $I \subset J$, with $\mathsf{supp}(I,\mathcal{D}) \leq \mathsf{supp}(J \setminus I,\mathcal{D})$. It can be shown that, after one modification, the minimal possible lift is either

$$\frac{|\mathcal{D}| \times (\mathsf{supp}(J,\mathcal{D}) - 1)}{\mathsf{supp}(I,\mathcal{D}) \times (\mathsf{supp}(J \setminus I,\mathcal{D}) - 1)}$$

or

$$\frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{(\mathsf{supp}(I,\mathcal{D}) + 1) \times \mathsf{supp}(J \setminus I,\mathcal{D})}.$$

Which case is minimal depends on the ratio of the supports of the itemsets I, $J \setminus I$ and J. Nevertheless, given these supports we can return the smallest of the two as minimal possible lift after one modification and denote the result by $\mathsf{mpl}(J,I,1)$. To generalize this to an arbitrary number k of modifications and obtain $\mathsf{mpl}(J,I,k)$, we recursively repeat this computation k times, with updated supports of the itemsets I, J and $J \setminus I$. The crucial property of minimal possible lift is the following:

Proposition 5. *Let J, I as above. If $J \in \mathsf{FBI}(\mathcal{D}',\tau)$ for some $\mathcal{D}'$ obtained from $\mathcal{D}$ by at most k modifications, then $\mathsf{mpl}(J,I,k) \leq \tau$.* □

Hence, if $\mathbb{A}$ consists of all itemsets J for which $\mathsf{mpl}(J,I,k) \leq \tau$, for some $I \subset J$ such that $\mathsf{lift}(J,\mathcal{D}) = \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\mathsf{supp}(I,\mathcal{D}) \times \mathsf{supp}(J \setminus I,\mathcal{D})}$, then it is guaranteed that all itemsets in $\mathsf{FBI}(\mathcal{D}',\tau)$ are returned, as desired by property (†). Algorithm

A-FBIMINER mines all itemsets J for which $\mathsf{mpl}(J, I, k) \leq \tau$, for I as above, and thus returns $\mathbb{A}$.

▷ **Avoiding New Inconsistencies.** Before explaining how A-FBIMINER works, we first explain how the set $\mathbb{A}$ of almost forbidden itemsets can be used to guarantee clean repairs. We come back to the repair algorithm in Sect. VI-C. Consider a modification $o_{rep,i}$ of a dirty object o_i. Our repair algorithm rejects this modification in the following cases:

— *Old inconsistency*: These are itemsets which should not be present in the repaired dataset, as they are already known to be inconsistent in $\mathcal{D}$. This happens if $o_{rep,i}$ covers an itemset $C \in \mathbb{A} \cap \mathsf{FBI}(\mathcal{D}, \tau)$. It also happens if $o_{rep,i}$ covers an itemset $C \in \mathbb{A}$ with $\mathsf{supp}(C, \mathcal{D}) = 0$, which are itemsets that do not occur in the original dataset, but would be forbidden if they did occur. Indeed, an incautious repair could introduce such an itemset in the "repaired" dataset. In other words, no itemset should be present in $o_{rep,i}$ that is already known to be forbidden. We denote this set as $\mathbb{A}_{old}$.

Repair Safety: Old inconsistencies are avoided when a repair $o_{rep,i}$ does not support any itemsets in $\mathbb{A}_{old}$.

—*Potential inconsistency*: Object $o_{rep,i}$ covers an itemset $C \in \mathbb{A}$ with $\mathsf{lift}(C, \mathcal{D}) > \tau$ and $\mathsf{lift}(C, \mathcal{D}'_i) < \mathsf{lift}(C, \mathcal{D})$, where $\mathcal{D}'_i$ is $\mathcal{D}$ in which only o_i is replaced by $o_{rep,i}$ and all other objects in $\mathcal{D}$ are preserved. In other words, when $o_{rep,i}$ covers an almost τ-forbidden itemset that is not yet τ-forbidden in $\mathcal{D}$, modifications that reduce the lift of this itemset should be prevented. We denote this set as $\mathbb{A}_{pot}$.

Repair Safety: Since it is infeasible to recompute the lift of all itemsets in $\mathbb{A}_{pot}$, we opt for a more efficient method which suffices to ensure that the lift of the itemsets $I \in \mathbb{A}_{pot}$ doesn't drop, by asserting that (i) no occurrence of I has been removed (which would decrease the numerator in its lift) and (ii) no occurrence of a strict subset of I has been added (which would increase the denominator in its lift). By guaranteeing that $\mathsf{supp}(I, \mathcal{D}) \geq \mathsf{supp}(I, \mathcal{D}')$ and for all $J \subsetneq I : \mathsf{supp}(J, \mathcal{D}) \leq \mathsf{supp}(J, \mathcal{D}')$, it follows that $\mathsf{lift}(I, \mathcal{D}') \geq \mathsf{lift}(I, \mathcal{D})$.

It can be shown that these two safety checks suffice to guarantee that no forbidden itemsets will be found in accepted repairs. We declare a candidate repair to be safe if it passes both checks.

B. Mining Almost Forbidden Itemsets

We now describe algorithm A-FBIMINER. It is similar to FBIMINER, except for the relaxed lift measure and different pruning strategies. Recall that algorithm A-FBIMINER is to mine almost forbidden itemsets without looking at repairs, i.e., only $\mathcal{D}$ and an upper bound k on the number of modified objects is available. Clearly k is at most $|\mathcal{D}_{dirty}|$. To adapt the pruning strategies from FBIMiner, the underlying properties must be revised to take possible modifications into account. Since clean objects are never modified, a tight bound on the support of an itemset in any $\mathcal{D}'$, obtained from $\mathcal{D}$ by at most k modifications, can be obtained. Indeed, observe that for any itemset I, the following holds:

$$\mathsf{supp}(I, \mathcal{D}_{clean}) \leq \mathsf{supp}(I, \mathcal{D}') \leq \mathsf{supp}(I, \mathcal{D}_{clean}) + k$$

An immediate consequence is that A-FBIMINER must also consider itemsets I with $\mathsf{supp}(I, \mathcal{D}_{clean}) = 0$ as these can become supported in repairs. Furthermore, we can now modify Propositions 1 and 2:

Proposition 6. *For any two itemsets I and J such that $I \subset J$, if J is a τ-forbidden itemset in $\mathcal{D}'$,then we have that* $\mathsf{supp}(I, \mathcal{D}_{clean}) \geq \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau} - k$. □

Proposition 7. *For any three itemsets I, J and K such that $I \subset J \subset K$, if K is a τ-forbidden itemset in $\mathcal{D}'$, then* $\mathsf{supp}(I, \mathcal{D}_{clean}) - \mathsf{supp}(J, \mathcal{D}_{clean}) \geq \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} - k$. □

Similarly to the pruning strategies for FBIMiner, we again use the trivial lower bound $\mathsf{supp}(J, \mathcal{D}') = 1$. Furthermore, to make use of these propositions for pruning, note that we do not know $\sigma^{max}_{I,\mathcal{D}'}$. Indeed, recall that we do not consider any particular repair $\mathcal{D}'$. Instead, $\sigma^{max}_{I,\mathcal{D}_{clean}}$ can be computed, and it holds that $\sigma^{max}_{I,\mathcal{D}'} \leqslant \sigma^{max}_{I,\mathcal{D}_{clean}} + k$. As a consequence, Prop. 6 is used to prune supersets of I whenever $\mathsf{supp}(I, \mathcal{D}_{clean}) < \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_{clean}} + k) \times \tau} - k$.

From Prop. 7, it follows that non-generator pruning can be applied to an itemset I if $\frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} > k$. Observe that $0 < \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} \leqslant 1$ and hence the impact of this ratio is almost negligible. Therefore, instead of computing $\sigma^{max}_{I,\mathcal{D}'}$ for every I, we use an estimate for $\sigma^{max}_{\emptyset,\mathcal{D}'}$ instead, i.e., $\sigma^{max}_{\emptyset,\mathcal{D}clean} + k$. Prop. 7 implies that $\mathsf{supp}(I, \mathcal{D}_{clean}) - \mathsf{supp}(J, \mathcal{D}_{clean}) \geq \frac{1}{\tau} - \frac{(\sigma^{max}_{\emptyset,\mathcal{D}_{clean}} + k)}{|\mathcal{D}|} - k$ must hold for every $I \subseteq J$ for which J is τ-forbidden in any $\mathcal{D}'$ obtained by using k modifications.

Finally, Prop. 3 also needs to be adapted to account for possible modifications. Since the mpl measure depends on the ratio of $\mathsf{supp}(J, \mathcal{D}')$ and its partitions, it does not preserve the anti-monotonicity of the lift denominator. Instead, we need to compute the worst case increase in the denominator of the lift measure:

Proposition 8. *For any two itemsets I and J such that $I \subset J$, it holds that* $\mathsf{lift}(J, \mathcal{D}') \geq$

$$\frac{\mathsf{supp}(J, \mathcal{D}') \times |\mathcal{D}|}{\min_{S \subset I} \big\{ (\mathsf{supp}(S, \mathcal{D}_{clean}) + k) \times (\mathsf{supp}(I \setminus S, \mathcal{D}_{clean}) + k) \big\}}.$$ □

As before, we use this proposition for $\mathsf{supp}(J, \mathcal{D}') = 1$. These three properties allow substantial pruning when mining almost forbidden itemsets similarly as explained for FBIMINER.

▷ **Batch execution.** Propositions 6 and 7 also show that the number k of modifications considered has a direct impact on the pruning capabilities of algorithm A-FBIMINER. Indeed, suppose that k takes the maximal value, i.e., $k = |\mathcal{D}_{dirty}|$. Then, it is likely that the bounds given in these propositions do not allow for any pruning. Worse still, the minimal possible lift, which also depends on k, will declare many itemsets to be almost forbidden. Although A-FBIMINER will need to be run only once to obtain the set $\mathbb{A}$ and all dirty objects can be repaired based on $\mathbb{A}$, this set will be big and inefficient to compute (due to lack of pruning). On the other hand, when $k = 1$, pruning will be possible and $\mathcal{A}$ will be of reasonable size. However, to clean all dirty objects, we need to deal with

them one-at-a-time, re-running A-FBIMINER for $k = 1$ after a single dirty object is repaired.

Between these two extremes, we propose to process the dirty objects in batches. More specifically, we partition $\mathcal{D}_{dirty}$ into blocks of a size r, optimizing the trade-off between the runtime of A-FBIMINER and the number of runs. The question, of course, is what block size to select. We already described block size $r = 1$ and $r = |\mathcal{D}_{dirty}|$. We next use Prop. 6 and Prop. 7 to identify ranges for r for which pruning may still be possible.

Let I and J be itemsets such that $I \subset J$. Proposition 6 is only applicable if $\frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau} > r$, and Prop. 7 is only applicable if $\frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} > r$, where $\mathcal{D}'$ is now any repair obtained from $\mathcal{D}$ by at most r modifications. It is easy to see that $\frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau} \geqslant \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|}$ and hence $r = \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau}$ is the maximal block size for which one can expect pruning. As before letting $\mathsf{supp}(J, \mathcal{D}') = 1$ and $I = \emptyset$, we identify $r = \frac{|\mathcal{D}|}{(\sigma^{max}_{\emptyset,\mathcal{D}_{clean}} + r) \times \tau}$ as a maximal block size that allows pruning based on Prop. 6. Additional pruning based on Prop. 7 is possible for lower block sizes, i.e., when $\frac{1}{\tau} - 1 > r$. Here we again use that $\frac{|\mathcal{D}|}{\sigma^{max}_{I,\mathcal{D}'}} \geq 1$. We hence also identify $r = \frac{1}{\tau} - 1$ as block size that allows substantial pruning.

Of course, there is no universally optimal block size r. The specifics of the data and even the choice of which objects to include in a partition of r objects all impact the pruning power. It is also important to note that the block sizes derived above guarantee that the associated propositions are *applicable*, but still offer *reduced pruning power* in comparison to smaller block sizes. In the experimental section, we show that $r = \frac{1}{2\tau}$ provides a sensible default value of r, whereas $r = \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau}$ improves the runtime on datasets with a small number of attributes.

C. Repair Algorithm

We are finally ready to describe our algorithm REPAIR, shown in Alg. 2. It takes as input the dirty and clean objects, $\mathcal{D}_{dirty}$ and $\mathcal{D}_{clean}$ respectively, a similarity function, and the lift threshold τ. The dirty objects $\mathcal{D}_{dirty}$ are partitioned in blocks R_i of size r. Per block, the set of almost forbidden itemsets $\mathbb{A}_i$ is computed. At this point, the repair process depends only on the two sets R_i and $\mathbb{A}_i$. After each processed block, $\mathcal{D}$ is updated with the found repairs in $\mathcal{D}'$, denoted as $\mathcal{D} \oplus \mathcal{D}'$. By default, the set $\mathcal{D}_{clean}$ is not altered during the entire repair process, ensuring that subsequent repairs are not based on previously imputed values, to avoid the propagation of undesirable repair choices.

For each dirty object in R_i, a candidate repair is generated by replacing some of its items by items from a similar, but clean object in $\mathcal{D}_{clean}$. By using the most similar objects to produce candidate repairs, we try to minimize the difference between $\mathcal{D}'$ and $\mathcal{D}$. This approach is in line with the commonly used *hot deck imputation* in statistical survey editing [4].

If the candidate repair is safe (as explained before) with respect to the almost forbidden itemsets in $\mathbb{A}_i$, then it is added to the set $\mathcal{D}'$ (line 12). Otherwise, the next candidate clean object is considered (loop line 8–13) until a repair is found (line 13) or all candidate repairs have been rejected. In this case, o_i is added to the set of unrepaired objects $\mathcal{D}''$ (line 14), for further user inspection.

Algorithm 2 Repairing dirty objects

```
1:  procedure REPAIR(D_dirty, D_clean, linsim, τ)
2:    for all R_i ∈ blocks(D_dirty, r) do
3:      r := |R_i|
4:      D' := ∅; D'' = ∅
5:      A_i = A-FBIMINER(D ⊕ D', r, τ)
6:      for all o_i ∈ R_i do
7:        success:=false
8:        for all o_c ∈ D_clean in sim(o_c, o_i) desc. order do
9:          o_rep,i = MODIFY(o_i, o_c)
10:         if SAFE(o_i, o_rep,i, A_i) then
11:           success := true
12:           D' := D' ∪ o_rep,i
13:           break
14:       if not success then D'' = D'' ∪ o_i
15:     return (D', D'')
```

It remains to explain how candidate repairs are generated. For each dirty object o_i, the algorithm consecutively processes the clean objects o_c in order of their similarity to o_i. The algorithm subsequently modifies the dirty object o_i by means of the procedure MODIFY(o_i, o_c) (line 8). The resulting object is denoted by $o_{rep,i}$. In our implementation, MODIFY(o_i, o_c) replaces items (A, v) in o_i by $(A, o_c[A])$ that occur in the τ-forbidden itemsets covered by o_i, i.e., only the items that are part of inconsistencies are modified.

VII. USER INTERACTION

The cleaning methods outlined in this paper have been designed such that they can be run fully automatically, without any user input or interaction. Of course, in practice, optional user input is often desired. Such a mechanism can readily be integrated into our method. Indeed, the repairing process relies only on the sets of forbidden and almost forbidden itemsets, $\mathsf{FBI}(\mathcal{D}, \tau)$ and $\mathbb{A}$, respectively. A user can validate the discovered itemsets, answering the simple question "are these items allowed to co-occur?". Itemsets that are rejected by the user can be discarded from the respective sets. The algorithms will then work as desired, considering the user-rejected (almost) forbidden itemsets to be semantically correct. Likewise, a user could be shown the top-k lowest lift forbidden itemsets and asked to confirm which itemsets to remove. The efficient discovery of such top-k itemsets and experimental evaluation of user interactions are left for future work.

VIII. EXPERIMENTS

In this section, we experimentally validate our proposed techniques, by answering the following questions:

- Does FBIMINER find a manageable set of errors efficiently? Are the forbidden itemsets actually errors? What is the impact of pruning?
- Can almost forbidden itemsets be mined efficiently? How does the block size impact runtime?

- Is the repair algorithm able to find low-cost repairs efficiently? How often is it impossible to repair an object?

A. Experimental Settings

The experiments were conducted on real-life datasets from the UCI repository (http://archive.ics.uci.edu/ml/). We show results for six datasets, their descriptions are given in Table II. The Adult database was preprocessed by discretizing ages and removing other continuous attributes. The algorithms have been implemented in C++, the source code and used datasets are available for research purposes [1]. The program has been tested on an Intel Xeon Processor (2.9GHZ) with 32GB of memory running Linux. Our algorithms run in main memory. Reported runtimes are an average over five independent runs.

Table II. STATISTICS OF THE UCI DATASETS USED IN THE EXPERIMENTS. WE REPORT THE NUMBER OF OBJECTS, DISTINCT ITEMS, AND ATTRIBUTES.

Dataset	$\|\mathcal{D}\|$	$\|\mathcal{I}\|$	$\|\mathcal{A}\|$
Adult	48842	202	11
CensusIncome	199524	235	12
CreditCard	30000	216	12
Ipums	70187	364	32
LetterRecognition	20000	282	17
Mushroom	8124	119	23

B. Forbidden Itemset Mining

We ran the forbidden itemset mining algorithm FBIMINER with full pruning, reporting the total runtime, the number of forbidden itemsets, and the number of objects containing a forbidden itemset, for increasing values of τ. For the larger datasets, Ipums and CensusIncome, a smaller τ range was considered. This prevents an explosion in the number of forbidden itemsets and the associated high runtime. The results are shown in Fig. 2, Fig. 3 and Fig. 4, respectively.

The results show that the runtime of the algorithm (Fig. 2) scales linearly with τ. As a result of the depth-first search, the runtime is strongly influenced by the number of distinct items. As a consequence, the algorithm runs slowest on the Ipums dataset. The runtime on the LetterRecognition dataset is explained by its relatively high number of items, and the fact that it contains many forbidden itemsets.

The number of forbidden itemsets (Fig. 3) is typically small, although there is a stronger than linear increase as the lift threshold increases, illustrating that τ should indeed be chosen very small. Especially for the LetterRecognition database, the number of forbidden itemsets increases exponentially. This occurs because the dataset is very noisy, since the contained letters were randomly distorted. In contrast, the less noisy Adult and CensusIncome datasets have relatively few dirty objects. The number of dirty objects (Fig. 4) naturally follows a similar pattern to the number of forbidden itemsets, with an occasionally big increase if a forbidden itemset with a relatively high support is discovered.

To answer the question "Are the forbidden itemsets actually errors?", a gold standard for the subjective task of data cleaning would be needed. Although synthetic error generators exist [38], [39], they require the constraints to be known up front. In line with [11], we therefore evaluate forbidden itemsets manually for usefulness, obtaining only a precision score. The results for the Adult dataset, which is the most readily interpretable, are shown in Table III. Precision is very high for small τ-values, and keeps up around 50% in the τ-range, which is a high score for an uninformed method. We believe a high precision is important to instill faith in an eventual user. Note that we report the precision of the *forbidden itemsets*, the number of erroneous objects is a multiple of this value, as illustrated by Fig. 4.

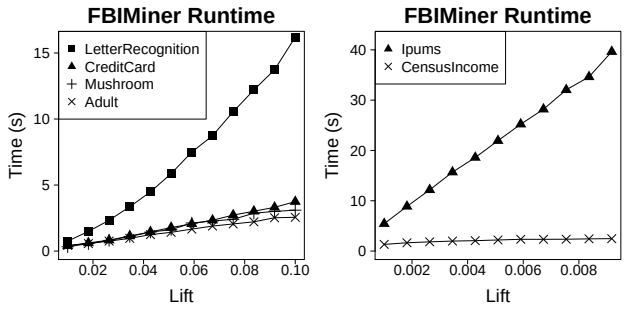

Figure 2. Runtime of FBIMiner in function of maximum lift threshold τ.

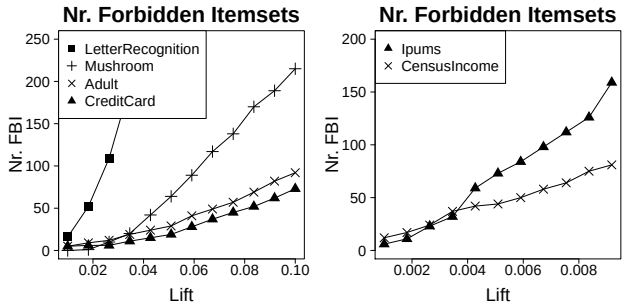

Figure 3. Number of Forbidden Itemsets in function of maximum lift threshold τ.

In order to evaluate the influence of the pruning strategies, we report the number of itemsets processed with only one type of pruning enabled, and contrast these with the number of itemsets processed when all pruning is enabled. We distinguish between *Min. Supp* pruning, using Prop. 1 on line 15 of Alg. 1; *Lift Denominator* pruning, using Prop. 3 on line 13; and *Support Diff* pruning, using Prop. 2 on line 20.

The results are shown in Fig. 5 for the Adult and CreditCard datasets; results for CensusIncome and LetterRecognition were similar. On the Mushroom and Ipums datasets, which have many attributes, FBIMiner became infeasible for larger values of τ without Support Diff pruning. Clearly, Support Diff pruning is dominant in most cases. Since this strategy also entails non-generator pruning, it is definitely crucial for the runtime of FBIMiner. Especially as τ increases, the other strategies also improve the overall result, indicating that all are beneficial and complementary to each other. We do not show the results when all pruning is disabled since all itemsets are then considered (independently of τ), leading to a high number of processed itemsets and running time.

A separate issue is the maximal frequency of a forbidden

[1] http://adrem.uantwerpen.be/joerirammelaere

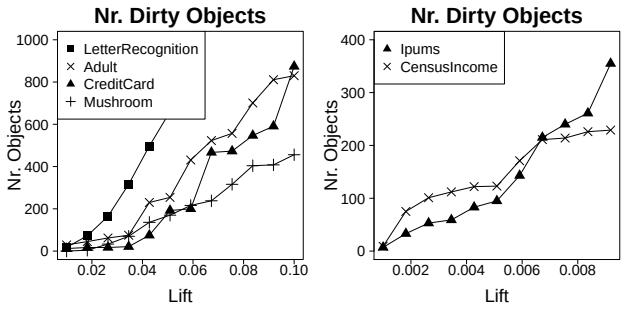

Figure 4. Number of objects containing one or more Forbidden Itemsets in function of maximum lift threshold τ.

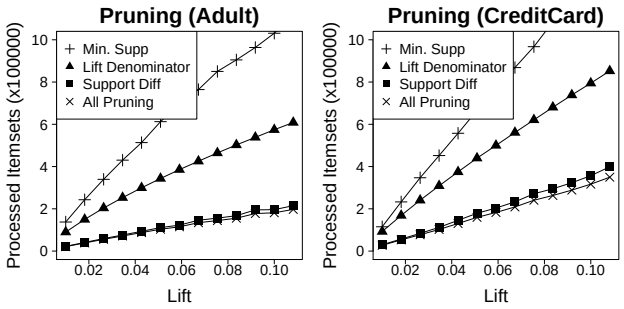

Figure 5. Number of itemsets processed with different pruning strategies in function of maximum lift threshold τ.

Table III. PRECISION OF DISCOVERED FORBIDDEN ITEMSETS ON ADULT DATASET.

	τ-value					
Dataset	0.01	0.026	0.043	0.067	0.084	0.1
Nr. FBI	5	12	24	49	69	92
Precision	100%	67%	71%	61%	55%	45%

Table IV. RUNTIME INFLUENCE OF MAXIMAL FREQUENCY BOUND IN FUNCTION OF τ. VALUES SHOWN ARE THE RUNTIME WITH FREQUENCY BOUND AS A PERCENTAGE OF RUNTIME WITHOUT FREQUENCY BOUND.

	τ-value					
Dataset	0.01	0.026	0.043	0.067	0.084	0.1
Adult	95%	92%	93%	98%	98%	112%
CreditCard	95%	89%	77%	96%	110%	97%

itemset, used on line 8 of Alg. 1. Recall that this is not a pruning strategy: using the frequency bound increased the number of itemsets processed, but may reduce runtime by avoiding certain unnecessary lift computations. This bound was disabled for the previous pruning results, to prevent painting a distorted picture of the influence of each pruning strategy. Table IV shows the runtime influence of the frequency bound as a percentage of the runtime without this bound. Results range from a 20% decrease to a 10% increase, and indicate that the frequency bound is typically beneficial for the runtime of FBIMiner, especially for smaller τ-values.

C. Almost Forbidden Itemsets

The discovery of almost forbidden itemsets is the most computationally expensive part in our methodology. Recall that the runtime of A-FBIMINER depends both on the lift threshold τ and the number of dirty objects as discovered by FBIMiner. Since a larger τ automatically entails a higher number of dirty objects, clearly scalability in τ is an issue.

For each dataset and each τ, we first run algorithm FBIMINER to obtain the forbidden itemsets. Let k denote the number of dirty objects found. We then run algorithm A-FBIMINER a number of times with block size r, indicating the number of dirty objects to be repaired at once, until all k objects have been repaired. Firstly, we report runtimes for the extreme cases of the block size, i.e., $r = 1$ and $r = k$. These runtimes are shown in Fig. 6a-6b for the first four datasets. Results for CensusIncome and Ipums were similar, the plots are deferred to the appendix of the full version [10].

The difference between both block sizes is clear. For $r = k$, runtimes start out reasonably low, but quickly explode as the algorithm loses its pruning power and computation becomes infeasible. This is the most problematic for Mushroom, which has a larger number of attributes, and LetterRecognition, which has a very high number of dirty objects k. Block size $r = 1$ remains feasible throughout the τ range, but is slower overall.

Next, we focus on the optimal block size r. As outlined in Sect. VI-B, we can identify the quantities $\frac{1}{\tau} - 1$ and $\frac{1}{\tau} \times \frac{|\mathcal{D}|}{\sigma^{max}_{\emptyset,\mathcal{D}'}}$ as the maximal block sizes for which Prop. 6 and Prop. 7, respectively, are still applicable. Fig. 6c-6d displays the obtained runtimes using these block sizes on the first four datasets, results for CensusIncome and Ipums are again omitted but similar. Note that the chosen values for r are τ-dependent. Consequently, for every τ-value, a different number of dirty objects is obtained and partitioned into blocks of a *different* absolute size. As expected, the runtimes are lower than for $k = 1$, while feasibility is better than for $r = k$, proving that the right block size indeed improves the overall performance of the algorithm. Runtime on the LetterRecognition dataset naturally suffers from the exponential increase in the number of dirty objects on that dataset. However, performance on the Mushroom dataset is still problematic: the number of attributes leads to a deep search tree, and pruning power is too limited. The same holds for Ipums. Plots of the obtained runtimes are deferred to the appendix of the full version [10].

As an alternative, we consider the block size $r = \frac{1}{2\tau}$, the halfway point between $r = 1$ and $r = \frac{1}{\tau} - 1$. Figure 6e-6f shows that this block size provides sufficient pruning power for the Mushroom dataset, and indeed outperforms all other considered sizes over the entire τ-range. For higher τ-values, the algorithm still struggles on the Ipums dataset, its high number of items proving problematic. On the other datasets, A-FBIMINER is fast for low values of τ, and feasible across the considered τ-range.

D. Data Repairing

In Table V we report on the quality of repairs obtained by algorithm REPAIR for various values of τ. The block size $r = \frac{1}{2\tau}$ was chosen, as described in the previous paragraph. We report the minimal and maximal similarity between a dirty object and its repair (within the τ range as above), with a similarity value of 1 indicating identical objects. The obtained repairs consistently have a high similarity in the given τ-range.

We also report the number of objects that could not be repaired at the highest τ-value, denoted as $\mathcal{D}''$. For Adult,

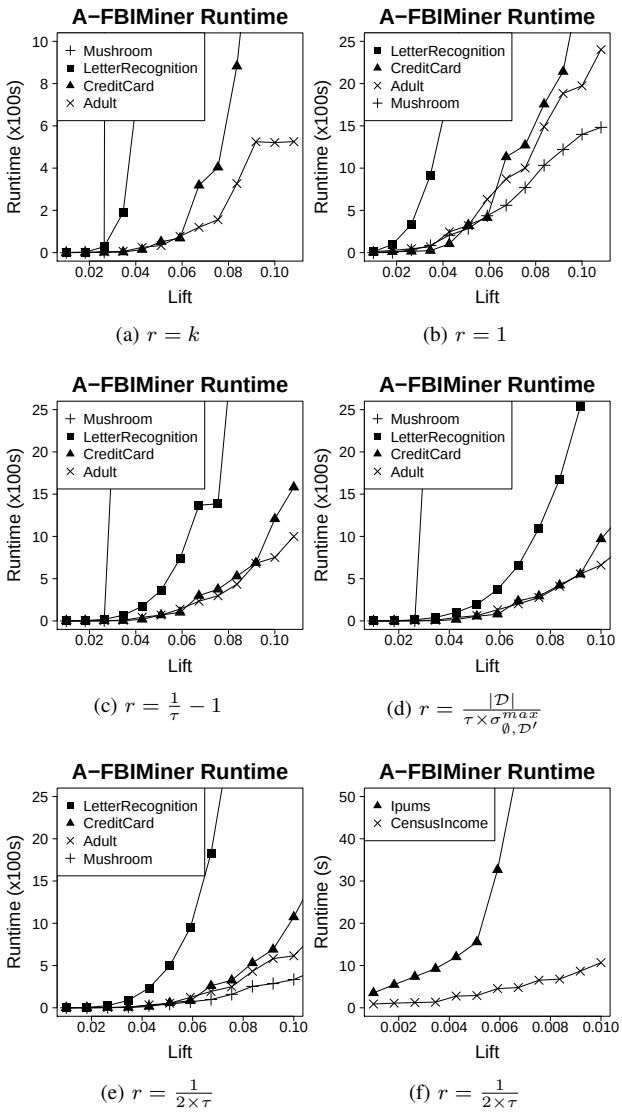

Figure 6. Runtime of A-FBIMINER in function of maximum lift threshold τ, for various block sizes r.

Clean	Age	Marital-status	Relationship	Sex	...
X	<18	Married-civ-spouse	Husband	Male	
V	>50	Married-civ-spouse	Husband	Male	
X	<18	Married-civ-spouse	Own-child	Female	
V	<18	Never-married	Own-child	Female	
X	22-30	Married-civ-spouse	Not-in-family	Male	
V	22-30	Never-married	Own-child	Male	
X	22-30	Married-civ-spouse	Wife	Male	
V	22-30	Married-civ-spouse	Wife	Female	
X	31-50	Married-civ-spouse	Wife	Male	
V	31-50	Married-civ-spouse	Husband	Male	

Figure 7. Example repairs on Adult dataset.

CensusIncome, CreditCard and LetterRecognition, only a few objects are unrepairable and this occurs only for high values of τ. A higher number of unrepairable objects is encountered for the Ipums and Mushroom datasets. This seems to suggest that a higher number of attributes causes problems for repairing.

Finally, Fig. 8 shows the runtime of algorithm REPAIR. The reported running times exclude the time needed for A-FBIMINER. Since the repair algorithm computes nearest

Table V. AVERAGE QUALITY OF REPAIRS.

Dataset	τ-range	Min-Max Sim.	$\lvert\mathcal{D}''\rvert$
Adult	0.01-0.1	0.94-0.95	1
CensusIncome	0.001-0.01	0.90-0.95	0
CreditCard	0.01-0.1	0.94-0.96	10
Ipums	0.001-0.01	0.95-0.98	94
LetterRecognition	0.01-0.1	0.96-0.98	33
Mushroom	0.01-0.1	0.94-0.99	238

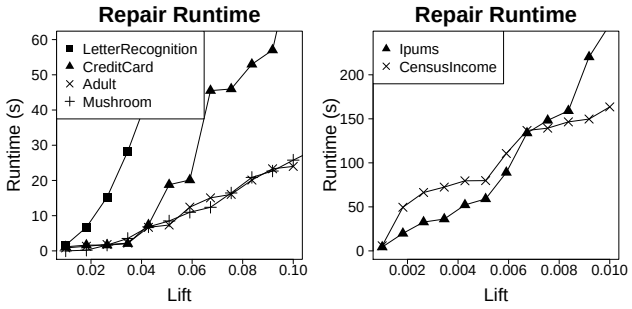

Figure 8. Runtime of the Repair algorithm in function of maximum lift threshold τ.

neighbors for all dirty objects, the runtime plots are similar in shape to the plots in Fig. 4. The required time to repair a single dirty object depends on the number of clean objects, which is typically close to $|\mathcal{D}|$. Note that the repair algorithm itself is independent of τ, which only affects FBIMINER and A-FBIMINER.

For illustrative purposes, Fig. 7 shows example repairs obtained on the Adult dataset ($\tau = 0.01$).

For our implementation, we make use of the lin-similarity measure [40] which weights both matches and mismatches based on the frequency of the actual values:

$$\mathsf{linsim}(o, o') = \frac{\sum_{A\in\mathcal{A}} S(o[A], o'[A])}{\sum_{A\in\mathcal{A}} \log(\mathsf{freq}(\{(A, o[A])\}, \mathcal{D})) + \log(\mathsf{freq}(\{(A, o'[A])\}, \mathcal{D}))}$$

where $S(o[A], o'[A])$ is given by

$$\begin{cases} 2\log(\mathsf{freq}(\{(A, o[A])\}, \mathcal{D})) & \text{if } o[A] = o'[A]; \text{ and} \\ 2\log(\mathsf{freq}(\{(A, o[A])\}, \mathcal{D})) + & \\ \log(\mathsf{freq}(\{(A, o'[A])\}, \mathcal{D})) & \text{otherwise.} \end{cases}$$

For example in the context of census data, a match or mismatch in gender would be more influential than a match or mismatch in the age category. Of course, any other similarity measure could be used instead. As part of future work, we intend to compare the influence of different similarity functions.

IX. CONCLUSION

We have argued that the classical point of view on data quality is too static, and proposed a general dynamic notion of cleanliness instead. We believe that this notion is quite interesting on its own and hope that it will be adopted and explored in various data quality settings.

In this paper, we have specialized the general setting by introducing so-called forbidden itemsets, established properties of the lift measure, and provided an algorithm to mine low-lift forbidden itemsets. Our experiments show that the algorithm

is efficient, and illustrate that forbidden itemsets capture inconsistencies with high precision, while providing a concise representation of dirtiness in data.

Furthermore, we have developed an efficient repair algorithm, guaranteeing that after repairs, no new inconsistencies can be found. By first mining almost forbidden itemsets, we assure that no itemsets become forbidden during a repair. This is an essential ingredient in our dynamic notion of data quality. Experiments show high quality repairs. Crucial here are our pruning strategies for mining almost forbidden itemsets.

As part of future work, we intend to experiment with different likeliness functions for the forbidden itemsets. Intuitively, the likeliness function for any type of single-object constraints. As long as the effect of a fixed number of modifications on this likeliness function can be bounded, then our approach remains applicable for larger classes of constraints. The repair algorithm also warrants further research: can a better repairability be achieved, especially on higher dimensional data? Finally, we seek to acquire datasets with ground truth such that an in-depth comparison of error precision and repair accuracy can be performed for different likeliness functions and repair strategies.

In conclusion, the dynamic view on data quality opens the way for revisiting data quality for other kinds of constraints or patterns. It would be interesting to see how to design repair algorithms in the setting for say, standard constraints such as conditional functional dependencies, among others. In addition, the impact of user interaction on the repairing process and the quality of repairs needs to be addressed.

REFERENCES

[1] W. Fan and F. Geerts, *Foundations of Data Quality Management*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[2] I. F. Ilyas and X. Chu, "Trends in cleaning relational data: Consistency and deduplication," *Foundations and Trends in Databases*, vol. 5, no. 4, pp. 281–393, 2015.

[3] I. P. Fellegi and D. Holt, "A systematic approach to automatic edit and imputation," *Journal of the American Statistical association*, vol. 71, no. 353, pp. 17–35, 1976.

[4] T. N. Herzog, F. J. Scheuren, and W. E. Winkler, *Data Quality and Record Linkage Techniques*. Springer, 2007.

[5] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Putting violations into context," 2013, pp. 458–469.

[6] ——, "Discovering denial constraints," *PVLDB*, vol. 6, no. 13, pp. 1498–1509, 2013.

[7] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren, "Curated databases," in *PODS*, 2008, pp. 1–12.

[8] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *VLDB*, vol. 3, no. 1-2, pp. 173–184, 2010.

[9] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The llunatic data-cleaning framework," *PVLDB*, vol. 6, no. 9, pp. 625–636, 2013.

[10] "Full version," http://adrem.uantwerpen.be/sites/adrem.uantwerpen.be/files/DataCleaningWithForbiddenItemsets-Full.pdf.

[11] F. Chiang and R. J. Miller, "Discovering data quality rules," *PVLDB*, vol. 1, no. 1, pp. 1166–1177, 2008.

[12] X. Chu, I. F. Ilyas, P. Papotti, and Y. Ye, "Ruleminer: Data quality rules discovery," in *ICDE*, 2014, pp. 1222–1225.

[13] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: an efficient algorithm for discovering functional and approximate dependencies," *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.

[14] F. Chiang and R. J. Miller, "A unified model for data and constraint repair," in *ICDE*. IEEE, 2011, pp. 446–457.

[15] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin, "On the relative trust between inconsistent data and inaccurate constraints," in *ICDE*. IEEE, 2013, pp. 541–552.

[16] M. Mazuran, E. Quintarelli, L. Tanca, and S. Ugolini, "Semi-automatic support for evolving functional dependencies," 2016.

[17] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *PVLDB*, vol. 5, no. 11, pp. 1483–1494, 2012.

[18] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller, "Continuous data cleaning," in *ICDE*. IEEE, 2014, pp. 244–255.

[19] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang, "Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples," in *SIGMOD*. ACM, 2016, pp. 893–907.

[20] T. Dasu and J. M. Loh, "Statistical distortion: Consequences of data cleaning," *PVLDB*, vol. 5, no. 11, pp. 1674–1683, 2012.

[21] C. C. Aggarwal, *Outlier analysis*. Springer, 2013.

[22] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys*, vol. 41, no. 3, 2009.

[23] M. Markou and S. Singh, "Novelty detection: a review," *Signal processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[24] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang, "Detecting data errors: Where are we and what needs to be done?" *PVLDB*, vol. 9, no. 12, pp. 993–1004, 2016.

[25] G. Webb and J. Vreeken, "Efficient discovery of the most interesting associations," *ACM Transactions on Knowledge Discovery from Data*, vol. 8, no. 3, pp. 1–31, 2014.

[26] J. Pei, A. K. Tung, and J. Han, "Fault-tolerant frequent pattern mining: Problems and challenges." *DMKD*, vol. 1, p. 42, 2001.

[27] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar, "Quantitative evaluation of approximate frequent pattern mining algorithms," in *SIGKDD*. ACM, 2008, pp. 301–309.

[28] H. Mannila and H. Toivonen, "Levelwise search and borders of theories in knowledge discovery," *DMKD*, vol. 1, no. 3, pp. 241–258, 1997.

[29] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD*, 1993, pp. 207–216.

[30] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li *et al.*, "New algorithms for fast discovery of association rules." *KDD*, pp. 283–286, 1997.

[31] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Transactions on Database Systems*, vol. 33, no. 2, 2008.

[32] M. J. Zaki and W. Meira Jr, *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[33] R. Bertens, J. Vreeken, and A. Siebes, "Beauty and brains: Detecting anomalous pattern co-occurrences," *arXiv preprint arXiv:1512.07048*, 2015.

[34] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *ICDT*, 1999, pp. 398–416.

[35] T. Calders and B. Goethals, "Depth-first non-derivable itemset mining," in *SDM*, 2005, pp. 250–261.

[36] L. Szathmary, P. Valtchev, A. Napoli, and R. Godin, "Efficient vertical mining of frequent closures and generators," in *Advances in Intelligent Data Analysis VIII*. Springer, 2009, pp. 393–404.

[37] M. J. Zaki and C.-J. Hsiao, "Charm: An efficient algorithm for closed itemset mining." in *SDM*, 2002, pp. 457–473.

[38] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro, "Messing up with bart: error generation for evaluating data-cleaning algorithms," *PVLDB*, vol. 9, no. 2, pp. 36–47, 2015.

[39] A. Arasu, R. Kaushik, and J. Li, "Datasynth: Generating synthetic data using declarative constraints," *PVLDB*, vol. 4, no. 12, 2011.

[40] S. Boriah, V. Chandola, and V. Kumar, "Similarity measures for categorical data: A comparative evaluation," in *SDM*, 2008, pp. 243–254.