Explaining Repaired Data with CFDs

Joeri Rammelaere
University of Antwerp
Antwerp, Belgium
joeri.rammelaere@uantwerpen.be

Floris Geerts
University of Antwerp
Antwerp, Belgium
floris.geerts@uantwerpen.be

ABSTRACT

Many popular data cleaning approaches are rule-based: Constraints are formulated in a logical framework, and data is considered dirty if constraints are violated. These constraints are often discovered from data, but to ascertain their validity, user verification is necessary. Since the full set of discovered constraints is typically too large for manual inspection, recent research integrates user feedback into the discovery process. We propose a different approach that employs user interaction *only at the start* of the algorithm: a user manually cleans a small set of dirty tuples, and we infer the constraint underlying those repairs, called an *explanation*. We make use of conditional functional dependencies (CFDs) as the constraint formalism. We introduce XPlode, an on-demand algorithm which discovers the best explanation for a given repair. Guided by this explanation, data can then be cleaned using state-of-the-art CFD-based cleaning algorithms. Experiments on synthetic and real-world datasets show that the best explanation can typically be inferred using a limited number of modifications. Moreover, XPlode is substantially faster than discovering all CFDs that hold on a dataset, and is robust to noise in the modifications.

PVLDB Reference Format:
Joeri Rammelaere and Floris Geerts. Explaining Repaired Data with CFDs. *PVLDB*, 11(11): 1387-1399, 2018.
DOI: https://doi.org/10.14778/3236187.3236193

1. INTRODUCTION

Dirty data is a significant problem for any sizeable organization. The amounts of data being processed have skyrocketed, and manual cleaning has long become infeasible. Research on data cleaning has provided methods for the automatic detection and repairing of various kinds of dirtiness. These methods are often rule-based: Constraints to which a database should adhere are encoded in some declarative constraint formalism, and repair algorithms modify the data such that no constraints are violated (see [23, 14] for an overview of rule-based approaches to data cleaning).

Proceedings of the VLDB Endowment, Vol. 11, No. 11

DOI: https://doi.org/10.14778/3236187.3236193

To use these repair methods, the challenge is then to obtain the correct constraints for repairing. Constraints can be discovered automatically from data [8, 16, 22, 31, 29, 34, 38, 19, 11], but such methods often discover an excessive amount of constraints. Many of these are spurious or invalid, and hence not suitable for actually repairing the data. To determine which constraints are *semantically valid* and *useful for repairing*, semantic knowledge of the data is required, which typically only a human user can provide.

We thus wish to harness the user's knowledge, while limiting the amount of interaction required. We therefore consider user interaction in the form of *manual repairs*. It is natural to assume that users know how certain errors *should be repaired*, based on expertise. Nevertheless, the *formal* constraints underlying these repair actions may be unknown to the user or hard to formulate precisely. Indeed, verifying candidate constraints may involve the inspection of the entire dataset to ensure that their violation sets coincide precisely with the errors in the data. The initial user effort can possibly be guided by high-precision error detection and data profiling methods [33, 3, 32]: even if such methods don't detect all errors, and cannot repair them, they can bootstrap the repairing process. We develop a method that generates constraints that are *consistent* with the repairs made by a user. We refer to such constraints as *explanations*. Intuitively, when such an explanation is used for *repairing*, the user provided repairs are left *intact*. As such, we maximally take the user input into account. The actual repairs, based on the explanations, can subsequently be performed using any state-of-the-art repair algorithm, such as those presented in [5, 13, 17, 18, 24, 26], among others.

In this paper we will focus on the problem of finding *the best possible explanation* for a given (partial) repair. Hence, our method should be seen as a single, core component in a *larger interactive data quality process* in which a *full* repair is gradually constructed – by interweaving manual repairs by a user and automatic repairs based on explanations. As more repaired tuples become available, the corresponding "best" explanations become more likely to be the *correct ones for repairing*, and better assistance can be offered to the user.

Methodology. As constraint formalism we use the class of *conditional functional dependencies* (CFDs) [15], widely used in data cleaning. CFDs are easy to interpret, yet more expressive than regular functional dependencies (FDs). Our proposed method, XPlode (for e**Xpl**anation **o**n **de**mand), extracts a single constraint, the *best explanation*, from a dirty dataset and an associated repair. We introduce a scoring function to assess the quality of an explanation, and

Table 1: Running example: A customers dataset.

TID	CC	AC	PN	NM	STR	CT	ZIP
1	01	908	1111111	Mike	Tree Ave.	~~LA~~ MH	07974
2	01	908	1111111	Rick	Tree Ave.	~~GLA~~ MH	07974
3	01	212	2222222	Joe	5th Ave	NYC	01202
4	01	908	2222222	Jim	Elm Str.	MH	07974
5	44	131	3333333	Ben	High St.	EDI	EH4 1DT
6	44	131	4444444	Ian	High St.	EDI	EH4 1DT
7	44	908	4444444	Ian	Port PI	MH	W1B 1JH
8	~~44~~ 01	131	2222222	Sean	3rd Str.	UN	01202

prove that the outcome of XPlode is equivalent to a "naive" method that first discovers all CFDs, and then uses post-processing to find the explanation with maximal score.

Our algorithm is "on-demand" in that it dynamically generates candidate explanations, hereby attempting to quickly discover the best possible explanation of the partial repair. A similar on-demand exploration was used in [20] to discover constant patterns that fit a single given CFD. We generalize the theoretical foundation of their algorithm, extending it to a larger class of evaluation functions, called loose anti-monotonic [6]. This class includes useful functions such as a minimum, maximum, or average, and our scoring function.

Underlying XPlode is a CFD discovery process that employs the user's modifications to navigate the search space of constraints, and quickly finds the best explanation. What differentiates XPlode from other constraint discovery methods that employ user responses to prune the search space [21, 35] is that we *only require modifications* as user input, and consider *general CFDs*. By contrast, in [21] a constant CFD discovery process is bootstrapped by only a *single* modification, followed by a number of questions to the user about the validity of the *constraints*. Similarly, in [35] FDs capturing errors are found by post-processing a set of approximate FDs. The post-processing again involves questions to the user about the validity of data and *constraints*. Our method only requires *positive* feedback in the form of correct modifications as opposed to a user frequently having to *invalidate* a candidate constraint. Hence, our method spends little time on constraints that are not suitable for repairing. Moreover, by considering all user feedback together, our method becomes more robust to small mistakes, instead of requiring every individual user interaction to be fully correct.

Motivating Example. For illustration, we borrow the running example from [16]. Table 1 shows a dirty version D_{dirty} and a clean version D_{rep} of the data. In the clean version, the three crossed out values are replaced by those next to it. In other words, a user is to repair D_{dirty} by changing the cities (CT) in $t_1[\mathsf{CT}]$ from "LA" to "MH" and in $t_2[\mathsf{CT}]$ from "GLA" to "MH", and the country code (CC) in $t_8[\mathsf{CC}]$ from 44 to 01. We assume that a user is faced with only the dirty data, and unable to exactly formalize the CFD that is supposed to hold. Nevertheless, based on experience or real-world knowledge, a user may be able to clean some tuples. We then wish to derive a constraint underlying the modifications made by the user. Since the obtained repair is assumed to be partial, a CFD discovery algorithm can only suggest a useful CFD by discovering approximate CFDs, i.e., CFDs that only partially hold. However, there are too many approximate CFDs for a user to inspect. Instead, we discover CFDs based on corrections made by the user.

For example, suppose that the user corrects the "error" in t_1 by restoring $t_1[\mathsf{CT}]$ back to its correct value of "MH". Suppose that an algorithm is at hand that only discovers CFDs that somehow "explain" this correction. Intuitively, this corresponds to the CFDs "becoming cleaner" in the repaired version. Two such FDs are $([\mathsf{ZIP}, \mathsf{AC}] \rightarrow \mathsf{CT})$ and $([\mathsf{AC}, \mathsf{CC}] \rightarrow \mathsf{CT})$. Indeed, if $t_1[\mathsf{CC}] =$"MH", then both FDs can be made to hold by removing a single tuple (t_2). Before the correction, two deletions were required. That is, these FDs have become cleaner. When considering CFDs as well, $([\mathsf{CC}, \mathsf{PN}] \rightarrow \mathsf{CT}, (01, 1111111, \mathrm{MH}))$ and $(\mathsf{NM} \rightarrow \mathsf{CT}, (\mathrm{Mike}, \mathrm{MH}))$, and many other CFDs become candidate explanations. Previous approaches would now ask multiple questions to the user, as to which of the candidate CFDs are (in)valid, until a semantically valid CFD is obtained. In this paper, we propose to let the user continue with repairing the dirty instance. For example, the user may decide to correct $t_8[\mathsf{CC}]$ back to 01. One can verify (as we did experimentally) that, for certain thresholds on support and confidence, as will be defined later, only the FD $\varphi = ([\mathsf{AC}, \mathsf{CC}] \rightarrow \mathsf{CT})$ can be related to the two modifications made by the user. This FD, saying that country code and area code uniquely determine city, is known to be semantically valid on this data. Moreover, φ can now be used to automatically correct $t_2[\mathsf{CT}]$ to "MH", using any CFD-based repair algorithm.

Summary of Contributions.

1. We formally define what it means for a CFD to *explain* a set of modifications. To differentiate between different explanations, we define a scoring function based on the number of explained modifications. (Section 3)

2. We design an algorithm, XPlode, that discovers the best explanation, i.e., the explanation of highest score. Moreover, XPlode is an *on-demand* algorithm that avoids a full exploration of the search space, when possible. To this aim, XPlode leverages an upper bound of the scoring function which is *loose anti-monotonic*. We discuss how XPlode can be modified to discover *multiple explanations*. (Section 4)

3. To further increase the efficiency of XPlode, we introduce an approximate scoring function that can be computed in time linear in the number of changes made by the user. By contrast, the actual scoring function has an inherent exponential dependency on the number of changes. (Section 5)

4. We experimentally show that our method can discover the correct CFD for repairing from only a small number of modifications, saving considerable user effort compared to manual validation of constraints ranked by baselines such as confidence. Moreover, XPlode is robust to noise in the modifications, i.e., mistakes made by the user, and outperforms a post-processing approach that discovers all explanations and then finds the highest scoring one. (Section 6)

2. PRELIMINARIES

We consider a relation schema R defined over a set $\mathcal{A}$ of k attributes, where each attribute $\mathsf{A}_i \in \mathcal{A}$ is associated with a domain $\mathsf{dom}(\mathsf{A}_i)$. A tuple t over R is simply an element of $\mathsf{dom}(\mathsf{A}_1) \times \cdots \times \mathsf{dom}(\mathsf{A}_k)$. A (database) instance D of R is a finite set of tuples over R. For a set X of attributes in $\mathcal{A}$ and tuple t over R, $t[\mathsf{X}]$ denotes the projection of that tuple on the attributes in X. We assume that each tuple $t \in D$ has a unique identifier tid, e.g., a natural number. We denote by $D[\mathsf{tid}]$ the tuple t in D with identifier tid.

A *conditional functional dependency* (CFD) [15] φ over R is a pair $(\mathsf{X} \rightarrow \mathsf{A}, t_p)$, where (i) X is a set of attributes in $\mathcal{A}$, and A is a single attribute in $\mathcal{A}$; (ii) $\mathsf{X} \rightarrow \mathsf{A}$ is a standard

functional dependency (FD); and (iii) t_p is a *pattern tuple* with attributes in X and A, where for each B in $\mathsf{X} \cup \{\mathsf{A}\}$, $t_p[\mathsf{B}]$ is either a constant 'b' in $\mathsf{dom}(\mathsf{B})$, or an unnamed variable '_'. A CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ in which $t_p[\mathsf{A}] = \_$ is called *variable*, otherwise it is called *constant*. For constant CFDs, $t_p[X]$ may be assumed to consist of constants only. An FD is a (variable) CFD with t_p consisting solely of variables '_'.

The semantics of a CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ on an instance D is defined as follows. A tuple $t \in D$ is said to *match* a pattern tuple t_p in attributes X, denoted by $t[\mathsf{X}] \asymp t_p[\mathsf{X}]$, if for all $\mathsf{B} \in \mathsf{X}$, either $t_p[\mathsf{B}] = \_$, or $t[\mathsf{B}] = t_p[\mathsf{B}]$. The tuple t *violates* a variable CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ if $t[\mathsf{X}] \asymp t_p[\mathsf{X}]$ and there exists another tuple t' in D such that $t[\mathsf{X}] = t'[\mathsf{X}]$ and $t[\mathsf{A}] \neq t'[\mathsf{A}]$. A tuple t *violates* a constant CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ if $t[\mathsf{X}] = t_p[\mathsf{X}]$ and $t[\mathsf{A}] \neq t_p[\mathsf{A}]$. The set of all tids of tuples in D that together violate a CFD φ is denoted by $\mathsf{VIO}(\varphi, D)$. If $\mathsf{VIO}(\varphi, D) = \emptyset$, then D *satisfies* φ, denoted by $D \models \varphi$.

The *support* of a CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ in D, denoted by $\mathsf{supp}(\varphi, D)$, is defined as the number of tuples $t \in D$ such that $t[\mathsf{X}] \asymp t_p[\mathsf{X}]$, i.e., the support is the number of tuples in D that match the pattern of φ on the attributes in X. In line with the commonly used notion of confidence for approximate FDs [25, 22], we define the *confidence* of a CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ in D as $\mathsf{conf}(\varphi, D) = 1 - \frac{|D'|}{\mathsf{supp}(\varphi, D)}$, where $D' \subseteq D$ is a minimal subset such that $D \setminus D' \models \varphi$. This definition is suited to variable CFDs, where the set $\mathsf{VIO}(\varphi, D)$ contains all tuples that *together* violate the CFD, but are not necessarily violations by themselves. For instance, if a violation set for a variable CFD contains two tuples with different A-values, the CFD can be made to hold by altering just one of the tuples. In other words, $|D'|$ is the minimum number of tuples that need to be altered or removed for φ to be satisfied. For a constant CFD, $|\mathsf{VIO}(\varphi, D)| = |D'|$, and hence this definition of confidence reduces to the standard confidence of an association rule. We observe that $\mathsf{conf}(\varphi, D) = 1$ means that $D \models \varphi$ and $\mathsf{conf}(\varphi, D) = 0$ means that all tuples matching $t_p[\mathsf{X}]$ need to be altered.

3. EXPLAINING REPAIRS

We here formalize the problem of discovering a *single* CFD that best explains an observed, possibly partial, repair of the data. Such repairs are represented by *modifications*. We describe what modifications are in Section 3.1. *Explaining* modifications in terms of CFDs, and differentiating between explanations based on a *scoring function*, is discussed in Section 3.2. Our problem statement is given in Section 3.3.

3.1 Modifications

We consider a setting in which no CFDs are provided. Instead, we have at our disposal two database instances, D_{rep} and D_{dirty}, where D_{rep} is obtained from the "dirty" instance D_{dirty} by applying a number of *modifications* to tuples. We assume that these instances have the same set of tids, such that tuples $D_{rep}[\mathsf{tid}]$ and $D_{dirty}[\mathsf{tid}]$ are both well-defined for every tid occurring in either instance. Every changed tuple in D_{rep} contains one or more modifications:

Definition 1 (Modification). A *modification* $\mathfrak{m}$ is a quadruple $\mathfrak{m} = (\mathsf{tid}, \mathsf{A}, v_d, v_c)$, where tid is the identifier of the tuple that is being changed, A is the attribute that is changed, v_d is the dirty value which was replaced, and v_c is the new, clean value, different from v_d. Given D_{dirty} and D_{rep}, a modification $\mathfrak{m} = (\mathsf{tid}, \mathsf{A}, v_d, v_c)$ is *consistent* with D_{dirty} and D_{rep} when, for tuples $s = D_{dirty}[\mathsf{tid}]$ and $t = D_{rep}[\mathsf{tid}]$, $s[\mathsf{A}] = v_d$ and $t[\mathsf{A}] = v_c$. □

Given D_{dirty} and D_{rep}, we denote by $\mathfrak{M}(D_{dirty}, D_{rep})$ the set of all modifications that are consistent with D_{dirty} and D_{rep}. Observe that $\mathfrak{M}$ can contain at most one modification for each tid and attribute. It should be clear that D_{dirty} and D_{rep} uniquely determine $\mathfrak{M}(D_{dirty}, D_{rep})$, as it is merely the "diff" of these two instances. We simply write $\mathfrak{M}$ whenever the dirty and modified instances are clear from the context.

Given a set of modifications $M \subseteq \mathfrak{M}(D_{dirty}, D_{rep})$, we denote by $D_{dirty} \oplus M$ the version of D_{dirty} on which the modifications in M are applied. Consequently, $D_{dirty} \oplus \{\emptyset\} = D_{dirty}$ and $D_{dirty} \oplus \mathfrak{M}(D_{dirty}, D_{rep}) = D_{rep}$. We let $\sigma_M(D)$ denote the set of tuples in D that are involved in a modification, i.e., whose tids occur in a modification in M. Furthermore, $\sigma^{\mathsf{tid}}_M(D)$ denotes the set of tids in $\sigma_M(D)$.

Example 1. *In our running example,* $\mathfrak{M}$ *consists of modifications* $\mathfrak{m}_1 = (1, \mathsf{CT}, \mathrm{LA}, \mathrm{MH})$, $\mathfrak{m}_2 = (2, \mathsf{CT}, \mathrm{GLA}, \mathrm{MH})$, *and* $\mathfrak{m}_3 = (8, \mathsf{CC}, 44, 01)$. *Both* $\sigma_{\mathfrak{M}}(D_{dirty})$ *and* $\sigma_{\mathfrak{M}}(D_{rep})$ *consist of tuples* t_1, t_2 *and* t_8, *with* tids 1, 2 *and* 8. ♢

3.2 Explaining a Repair

Suppose that we are given a repair, represented by a set of modifications. These modifications are made by a user without any assumed knowledge of CFDs that may be required to hold on the data, capturing the semantics of "clean data". As explained in the introduction, we want to recommend a CFD based on the current repair, such that this CFD may be used to detect further errors and suggest modifications.

(ε, δ)-CFDs. Intuitively, we relate candidate CFDs for "*explaining*" the current repair to D_{rep}, as this database instance is regarded to be cleaner than D_{dirty}. To narrow down such candidate CFDs, consider the following example.

Example 2. *Consider the partial repair in our running example corresponding to the single modification* $\mathfrak{m}_1 = (1, \mathsf{CT}, \mathrm{LA}, \mathrm{MH})$. *The CFD* $\varphi_1 = (\mathsf{NM} \to \mathsf{CT}, (\mathrm{Mike}, \mathrm{MH}))$ *could serve as an explanation for this modification as it is satisfied on the partial repair and relates to tuple* t_1. *It is unlikely, however, that this CFD is useful. Indeed,* φ_1 *is only supported by a single tuple* (t_1) *and does not relate at all to, for example, modification* $\mathfrak{m}_2 = (2, \mathsf{CT}, \mathrm{GLA}, \mathrm{MH})$ *that will be made by the user to tuple* t_2. *Consider next CFD* $\varphi_2 = (\emptyset \to \mathsf{CT}, (\mathrm{MH}))$, *stating that all cities should be MH. It is a well-supported CFD (its support is the entire database) and relates to* $\mathfrak{m}_1$ *and* $\mathfrak{m}_2$. *It may not be useful, however, for further cleaning of the data. Indeed,* φ_2 *has a very low confidence: more than half of the data violates this CFD and hence too many tuples may be flagged as dirty.* ♢

This example illustrates the need for ensuring that explanations have sufficient *support*, as this excludes CFDs that are only supported by remaining errors and noise in D_{rep}. At the same time, since D_{rep} is only a partial repair, we should focus on explanations that only hold *approximately* in D_{rep}. All combined, this motivates the following definition.

Definition 2 ((ε, δ)-CFD). A CFD φ on D_{rep} is an (ε, δ)-CFD if $\mathsf{conf}(\varphi, D_{rep}) \geq 1 - \varepsilon$ and $\mathsf{supp}(\varphi, D_{rep}) \geq \delta$. We denote by $\Sigma_{(\varepsilon,\delta)}(D_{rep})$ the set of all such (ε, δ)-CFDs. □

From here on, the set $\Sigma_{(\varepsilon,\delta)}(D_{rep})$ is our candidate set of CFDs for explaining repairs. But what does it mean to "explain a repair"? We next address this question.

M-Repair Explanations. Our definition for explanations is based on the following intuitive condition:

> "A CFD explains a repair if the repair improves the cleanliness of the data w.r.t. the CFD."

We formalize this intuition by imposing three natural conditions on CFDs φ in $\Sigma_{(\varepsilon,\delta)}(D_{rep})$. Let $\mathfrak{M} = \mathfrak{M}(D_{dirty}, D_{rep})$.

1. The confidence of φ in D_{rep} should have increased compared to its confidence in D_{dirty} as the result of the modifications made to D_{dirty}. Since a confidence of 1 means that φ is no longer violated, an increase in confidence brings φ closer to being satisfied in the repair. Note that if the confidence of φ increases, then φ was necessarily violated in D_{dirty}.

2. We require that at least one of the violations of φ in D_{dirty} has a tid occurring in $\sigma^{\mathsf{tid}}_{\mathfrak{M}}(D_{dirty})$, i.e., a tid of a modified tuple. This ensures that the increase in confidence, as required by the first condition, is the deliberate effect of resolving a violation of φ. Such a condition is necessary: in the running example, the CFD $(\mathsf{CC} \to \mathsf{PN}, (01, 2222222))$ is not violated on t_8 in the dirty data. However, modification $\mathfrak{m}_3$ in t_8 does increase the confidence of this CFD.

3. We require that φ is not violated in $\sigma_{\mathfrak{M}}(D_{rep})$, the part of the data that was specifically cleaned by the user. This ensures that if one uses φ later for further repairing, using any state-of-the-art CFD-based repairing algorithm, the tuples in $\sigma_{\mathfrak{M}}(D_{rep})$ will not be altered, i.e., they remain clean.

We next state these conditions more generally in terms of a set $M \subseteq \mathfrak{M}(D_{dirty}, D_{rep})$ of modifications:

Definition 3 (M-Repair Explanation). Consider instances D_{dirty} and D_{rep}, modifications $M \subseteq \mathfrak{M}(D_{dirty}, D_{rep})$. The CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ is an *M-repair explanation* if

1. $\mathsf{conf}(\varphi, D_{dirty} \oplus M) > \mathsf{conf}(\varphi, D_{dirty})$;
2. $\mathsf{VIO}(\varphi, D_{dirty}) \cap \sigma^{\mathsf{tid}}_{M}(D_{dirty})$ is not empty; and
3. $\mathsf{VIO}(\varphi, \sigma_M(D_{dirty} \oplus M))$ is empty.

We denote by $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{dirty} \oplus M)$ the set of M-repair explanations in $\Sigma_{(\varepsilon,\delta)}(D_{dirty} \oplus M)$. □

Of particular interest is the case when $M = \mathfrak{M}$ and hence $D_{dirty} \oplus M = D_{rep}$. In this case, we also call CFDs in $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{rep})$ *global explanations*. The set of global explanations, however, can be quite sizeable, as we will show in the experimental section (Section 6). Worse still, explanations in this set do not necessarily relate to all modifications in $\mathfrak{M}$. In fact, a CFD may be an explanation because of just one of many modifications, as is illustrated next.

Example 3. *When discovering global explanations in our example dataset, with $\varepsilon = 0.25$ and $\delta = 2$, there are 18 candidate CFDs. Among these, FD $([\mathsf{AC}, \mathsf{CC}] \to \mathsf{ZIP}, (\_, \_, \_))$ is a global explanation, for $\mathfrak{M} = \{\mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3\}$, yet it is only related to one modification, namely $\mathfrak{m}_3$.* ♢

In order to distinguish between such "good" and "bad" explanations, we need to strengthen the connection between explanations and modifications. Instead of enforcing extra conditions on global explanations, we assign to them a quality metric. Observe that in the previous example, the CFD $([\mathsf{AC}, \mathsf{CC}] \to \mathsf{ZIP}, (\_, \_, \_))$ is a global explanation, i.e., it is an M-repair explanation for the entire set of modifications $M = \mathfrak{M}$, thanks to $\mathfrak{m}_3$, but it is not an M-repair explanation for the subset $M = \{\mathfrak{m}_1, \mathfrak{m}_2\}$. We therefore introduce the concept of *locally* explaining modifications, on which we will base our quality metric.

Local Repair Explanations. Example 3 indicates that good global explanations should also explain the modifications involved *locally*. Intuitively, this means that the three conditions stated in Definition 3 should not only hold for the entire M, but also for subsets $M' \subseteq M$. Requiring these conditions to hold for all subsets of $\mathfrak{M}$ is too strong, however, as there may not exist any global explanations with this property. We start by defining what it means for a CFD to locally explain a set of modifications.

Definition 4 (Local Repair Explanation). Given a CFD φ in $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{rep})$, we say that φ *locally explains* a set $M \subseteq \mathfrak{M}$ if for every non-empty subset $M' \subseteq M$, φ is an M'-repair explanation. □

In other words, when a global explanation φ locally explains M, it explains all repairs $D_{dirty} \oplus M'$ that can be obtained from applying modifications $M' \subseteq M$ to D_{dirty}. That is, in any order in which the modifications in M are applied, the cleanliness of the data w.r.t. φ improves at every step.

This notion gives rise to the following quality metric. Let φ be a global explanation in $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{rep})$. Then,

$$\mathsf{score}(\varphi, \mathfrak{M}) := \max\{|M| \,|\, M \subseteq \mathfrak{M} \text{ and } \varphi \text{ locally explains } M\}.$$

If φ has a score close to $|\mathfrak{M}|$, then almost all modifications in $\mathfrak{M}$ are both globally and locally explained. We are thus interested in global explanations with a high score.

Example 4. *In the running example, the CFD $([\mathsf{CC}, \mathsf{AC}] \to \mathsf{ZIP}, (\_, \_, \_))$ has a score of 1, since it only explains modification $\mathfrak{m}_3$. Indeed, it is easily verified that $\mathfrak{m}_1$ and $\mathfrak{m}_2$ do not improve the confidence of this CFD. Moreover, if a user would only supply $\mathfrak{m}_1$ and $\mathfrak{m}_3$, this CFD could not be used to automatically clean the remainder of the data (i.e., apply $\mathfrak{m}_2$). On the other hand, the CFD $([\mathsf{CC}, \mathsf{AC}] \to \mathsf{CT}, (\_, \_, \_))$ can explain all 3 modifications, leading to a perfect score of 3. Even if only $\mathfrak{m}_1$ and $\mathfrak{m}_3$ were supplied by the user, this CFD would have the highest score of 2, and could automatically apply $\mathfrak{m}_2$. This example strengthens our argument that user-supplied modifications can guide the CFD discovery process towards the CFD that is most useful for cleaning the remainder of the data.* ♢

3.3 Problem Statement

We now have all ingredients for our problem statement:

PROBLEM: Repair Explanation Discovery

INPUT: Instances D_{dirty} and D_{rep}, modifications $\mathfrak{M}(D_{dirty}, D_{rep})$, thresholds ε and δ.

OUTPUT: A global explanation $\varphi \in \mathsf{Explain}_{(\varepsilon,\delta)}(D_{rep})$ such that $\mathsf{score}(\varphi, \mathfrak{M})$ is maximal.

In other words, we want to find the global explanation that also locally explains the largest subset of modifications of $\mathfrak{M}$. Any solution to this problem has somehow embedded in it the problem of discovering CFDs. Lattice traversal algorithms for CFD discovery [16, 8], exhibit an inherent exponential dependency in $|\mathcal{A}|d$, where $|\mathcal{A}|$ is the number of attributes and d is the maximum number of values occurring in any attribute. We will show that testing whether or not candidate CFDs satisfy the support and confidence thresholds and are global explanations imposes minimal overhead on the overall CFD discovery process. The score computation,

however, has a severe impact on the performance. Indeed, for each candidate global explanation it requires (worst case) to consider all subsets of $\mathfrak{M}$. Not all is lost, however. We will show in Section 5, that an efficient and good approximation of the scoring function can be computed.

4. DISCOVERING REPAIR EXPLANATIONS

We next describe an algorithm for discovering the best repair explanation, as defined in our problem statement. A solution to this problem is to first discover all CFDs which globally explain the observed repair, and then return the CFD with the highest score in a post-processing step. Clearly, this method does a lot of unnecessary work. Instead, we present an on-demand algorithm XPLODE, which returns the best explanation as soon as it is known. We begin in Section 4.1 with a detailed overview of XPLODE. Correctness of XPLODE is shown in Section 4.2, contingent on the availability of a *loose anti-monotonic* upper bound function on the scores of certain sets of CFDs. Examples of such upper bounds are provided as well. In Section 4.3, we briefly discuss how to modify XPLODE for returning multiple CFDs. Implementation details are presented in Section 4.4.

4.1 XPlode: Explanations On-Demand

We first provide a detailed overview of algorithm XPLODE (for e**XPl**anations **o**n-**de**mand) and refer to Algorithm 1 for its pseudo-code. At the core of XPLODE is a traversal of a lattice, commonly used in FD and CFD discovery algorithms, in combination with the use of equivalence partitions to check whether CFDs are global explanations. These partitions, also used in FD and CFD discovery algorithms such as TANE [22] and CTANE [16], respectively, are well suited for the (ε, δ)-thresholds that we impose on confidence and support. We elaborate further on this partitioning technique in Section 4.4. For now, we assume that we can efficiently check whether or not a CFD φ is in $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{dirty} \oplus \mathfrak{M})$ for a given instance D_{dirty} and set $\mathfrak{M}$ of modifications, and that $\mathsf{score}(\varphi, \mathfrak{M})$ can be computed easily.

The description below focuses instead on how the best explanation can be found *during* the exploration of the lattice, rather than by post-processing $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{dirty} \oplus \mathfrak{M})$. The challenge is to quickly pinpoint a CFD in $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{dirty} \oplus \mathfrak{M})$ with guaranteed highest score.

To explain the workings of XPLODE, we need to introduce some concepts. First of all, the lattice that we traverse is the *power set lattice* of all attribute/value combinations (including the wildcard '_' as a special value). As in CTANE [16], we denote the lattice elements by (X, t_p), where X is a set of attributes and t_p is a pattern tuple over X. Hence, $(\mathsf{X}, t_p) \subseteq (\mathsf{Y}, s_p)$ iff $\mathsf{X} \subseteq \mathsf{Y}$ and $t_p = s_p[\mathsf{X}]$. In this case, we also say that (Y, s_p) is a *child* of (X, t_p) and the children of (X, t_p) are thus obtained by expanding (X, t_p) with all possible attribute/value pairs (A, a) where $\mathsf{A} \notin \mathsf{X}$ and $a \in \mathsf{dom}(\mathsf{A}) \cup \{\_\}$. Furthermore, since we are only interested in CFDs of high support, it suffices to only consider pairs (A, a) that have sufficient support in D_{rep}. That is, the number of tuples $t \in D_{rep}$ such that $t[\mathsf{A}] \asymp a$ should exceed δ.

Second, as in CTANE, an element (X, t_p) represents a set of *candidate* CFDs, denoted by $\mathsf{CandCFD}(\mathsf{X}, t_p)$, consisting of all CFDs of the form $(\mathsf{X} \setminus \{\mathsf{A}\} \to \mathsf{A}, t_p)$, for $\mathsf{A} \in \mathsf{X}$.

Finally, let $\mathsf{UB}(\mathsf{X}, t_p)$ be an upper bound on $\mathsf{score}(\varphi, \mathfrak{M})$ for any $\varphi \in \mathsf{CandCFD}(\mathsf{X}, t_p)$. As will be explained shortly, this upper bound function is used as a guide for the traversal through the lattice, and serves to ensure that the highest-score global explanation can be identified *without the need for exploring the full lattice.*

Algorithm XPLODE relies on a traversal of the lattice such that its elements (X, t_p) are visited in *descending order* according to their upper bound $\mathsf{UB}(\mathsf{X}, t_p)$. To this aim, we keep generated elements in a priority queue Φ, initially containing $(\emptyset, \emptyset)$ with upper bound $+\infty$ (line 3). During the run of the algorithm, we also maintain the global explanation $\varphi_{\max}$ with highest score seen so far, denoting its score by "max". Initially, $\varphi_{\max}$ is set to nil and $\mathsf{max} = 0$ (line 4). The algorithm ensures that, at any time, the queue Φ will only consist of elements (X, t_p) such that $\mathsf{UB}(\mathsf{X}, t_p) > \mathsf{max}$. In other words, we only visit elements if a better explanation can possibly be found among its candidate CFDs.

Suppose XPLODE is currently exploring element (X, t_p), i.e., the foremost element in the queue Φ with highest upper bound (line 6). For the current element (X, t_p), a CFD φ in $\mathsf{CandCFD}(\mathsf{X}, t_p)$ is selected, that (i) is a global explanation; and (ii) has highest score among all other global explanations in $\mathsf{CandCFD}(\mathsf{X}, t_p)$, if such a CFD exists (line 7). The score of φ, and the scores of all CFDs in $\mathsf{CandCFD}(\mathsf{X}, t_p)$ have already been computed at an earlier stage of the algorithm, when (X, t_p) was generated. We expand on this later.

If φ exists and $\mathsf{score}(\varphi, \mathfrak{M}) > \mathsf{max}$, then φ is a better explanation than $\varphi_{\max}$. In this case, $\varphi_{\max}$ is set to φ and max to $\mathsf{score}(\varphi, \mathfrak{M})$ (line 9). Furthermore, the queue Φ is updated by removing each element with an upper bound smaller than or equal to the new max-value (line 10). This guarantees that all elements in Φ have an upper bound larger than the current max-value, as pointed out previously.

Finally, all children (Y, s_p) of (X, t_p) are generated and the scores of all candidate CFDs in $\mathsf{CandCFD}(\mathsf{Y}, s_p)$ are computed (lines 11-13). We can thus indeed assume, as we did earlier, that when XPLODE considers (Y, s_p) at a later stage, all scores of its candidate CFDs are available. Furthermore, if $\mathsf{UB}(\mathsf{Y}, s_p) > \mathsf{max}$, then (Y, s_p) is inserted in Φ with upper bound value $\mathsf{UB}(\mathsf{Y}, s_p)$ (line 15). This guarantees that Φ contains only elements with a sufficiently high upper bound.

When the queue is empty, the algorithm terminates by returning $\varphi_{\max}$ (line 16). If $\varphi_{\max} \neq \mathsf{nil}$, the CFD $\varphi_{\max}$ is guaranteed to be a global explanation. In the next section, we identify sufficient conditions on the upper bound function such that $\varphi_{\max}$ is a global explanation with *maximal score.*

Remarks. (1) Multiple elements in Φ may hold the same maximal UB-value. We break ties by prioritizing on elements (X, t_p) with the highest-score CFD in $\mathsf{CandCFD}(\mathsf{X}, t_p)$. If ties persist, we pick (X, t_p) containing the highest number of wildcards in t_p. These choices can lead XPLODE quicker to elements that represent the best global explanation. (2) One may wonder why, when generating a child (Y, s_p) of (X, t_p), we compute the scores of all its candidate CFDs (lines 11-13), and do not limit the score computation to candidate CFDs that are in fact global explanations. Indeed, CFDs that do not globally explain repairs do not impact the final result and consequently, their score could be simply set to 0. This would result in avoiding potentially expensive score computations. However, since scores are used for tie-breaking, experiments show that it is more efficient if we do compute all scores. Intuitively, more scores lead more quickly towards elements with higher scores, leading to a swifter discovery of an explanation with maximal score.

Algorithm 1 On-demand algorithm XPLODE for obtaining the best explanation for a set $\mathfrak{M}$ of modifications.

procedure XPLODE(D_{dirty}, D_{rep}, $\mathfrak{M}$, ε, δ, $\mathsf{score}(\cdot)$, $\mathsf{UB}(\cdot)$)
  $\Phi \leftarrow$ PRORITYQUEUE($\{\}$)
  Insert $(\emptyset, \emptyset)$ into Φ with upper bound $+\infty$
  $\varphi_{\max} \leftarrow \mathsf{nil}$, $\mathsf{max} \leftarrow 0$
  while Φ is not empty **do**
    $(\mathsf{X}, t_p) \leftarrow$ POP(Φ)
    Let $\varphi \in \mathsf{CandCFD}(\mathsf{X}, t_p)$ such that φ is a global explanation with highest score among all the global explanations in $\mathsf{CandCFD}(\mathsf{X}, t_p)$
    if φ exists and $\mathsf{score}(\varphi, \mathfrak{M}) > \mathsf{max}$ **then**
      $\varphi_{\max} \leftarrow \varphi$, $\mathsf{max} \leftarrow \mathsf{score}(\varphi, \mathfrak{M})$
      Delete from Φ all elements with UB-value $\leq \mathsf{max}$
    for all children (Y, s_p) of (X, t_p) **do**
      for all $\psi \in \mathsf{CandCFD}(\mathsf{Y}, s_p)$ **do**
        Compute $\mathsf{score}(\psi, \mathfrak{M})$
      if $\mathsf{UB}(\mathsf{Y}, s_p) > \mathsf{max}$ **then**
        Insert (Y, s_p) into Φ with value $\mathsf{UB}(\mathsf{Y}, s_p)$.
  return $\varphi_{\max}$.

4.2 Correctness and Upper Bound Functions

We next show how to guarantee that when XPLODE outputs a global explanation, it is a global explanation of maximal score. The correctness of XPLODE entirely relies on the upper bound function $\mathsf{UB}(\cdot)$, as this function determines which elements are in the priority queue, and in what order. We first identify sufficient conditions on $\mathsf{UB}(\cdot)$ to guarantee correctness. Examples of "good" upper bound functions are described at the end of this section.

Correctness. XPLODE returns $\varphi_{\max}$ when the priority queue Φ is empty. Clearly, every element that was ever generated has either been visited, or is not in the queue because its UB-value is below the score max of $\varphi_{\max}$. Observe that for any element (X, t_p), its upper bound $\mathsf{UB}(\mathsf{X}, t_p)$ is larger than the score of any candidate CFD of (X, t_p). This implies that none of the elements generated during execution, have a candidate CFD with a score higher than max.

Correctness of XPLODE then requires that, when $\varphi_{\max}$ is returned as the best global explanation, then any element (X, t_p) in the lattice whose upper bound value is larger than max must have been added to the queue at some prior stage. To this aim, we require that the upper bound function $\mathsf{UB}(\cdot)$ is *loose anti-monotonic* [6]: For any (X, t_p) there exists a parent $(\mathsf{Y}, s_p) = (\mathsf{X} \setminus \{\mathsf{B}\}, t_p[\mathsf{X} \setminus \{\mathsf{B}\}])$ such that $\mathsf{UB}(\mathsf{X}, t_p) \leq \mathsf{UB}(\mathsf{Y}, s_p)$. That is, every element in the lattice has at least one parent with a higher or equal UB-value. The proof of correctness is deferred to the online appendix [1].

PROPOSITION 1. On input D_{dirty}, D_{rep}, $\mathfrak{M}$, ε, δ, $\mathsf{score}(\cdot)$, and $\mathsf{UB}(\cdot)$, the algorithm XPLODE returns the global explanation with maximal score, if it exists, provided that $\mathsf{UB}(\cdot)$ is loose anti-monotonic and for any element (X, t_p), $\mathsf{UB}(\mathsf{X}, t_p)$ is larger than the score of any of its candidate CFDs. □

Loose Anti-monotonic Upper Bounds. The question is now whether there exist non-trivial [1] upper bound functions that satisfy the conditions in Proposition 1. We answer this affirmatively in this section. Defining $\mathsf{UB}(\mathsf{X}, t_p)$ as the maximal *score* of the candidate CFDs of (X, t_p) does not suffice, however, as the following example illustrates.

[1]One can choose a constant function $\mathsf{UB}(\cdot)$. XPLODE then performs an exhaustive breadth-first lattice traversal.

EXAMPLE 5. *We return to the running example, and consider the CFD* $\varphi = (\mathsf{CC} \rightarrow \mathsf{CT}, (\_, \_))$*, which locally explains all modifications* $\{\mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3\}$*. Hence, its score is* 3 *(but it is not sufficiently confident, for* $\varepsilon = 0.25$*, to be a global explanation). This is a candidate CFD for the element* $(\{\mathsf{CC}, \mathsf{CT}\}, (\_, \_))$ *in the lattice, with parents* $(\mathsf{CC}, \_)$ *and* $(\mathsf{CT}, \_)$*. However, the CFDs* $\varphi_1 = (\emptyset \rightarrow \mathsf{CT}, (\_))$ *and* $\varphi_2 = (\emptyset \rightarrow \mathsf{CC}, (\_))$ *only have scores of* 2 *and* 1*, respectively. Indeed,* φ_1 *locally explains* $\{\mathfrak{m}_1, \mathfrak{m}_2\}$ *and* φ_2 *locally explains* $\mathfrak{m}_3$*. No larger sets are locally explained by these CFDs.* ♢

Instead, we observe the following. For a CFD φ, we define $\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})$ as the set of modifications in $\mathfrak{M}$ that apply to tuples in $\mathsf{VIO}(\varphi, D_{dirty})$. Here, a modification $\mathfrak{m}$ applies to a tuple t when they share the same tid-value. It follows directly from Definition 4 that the set of modifications that a global explanation can also locally explain, consists at most of those modifications involved in violations of that explanation in the dirty database instance. Hence, $\mathsf{score}(\varphi, \mathfrak{M}) \leq |\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})|$. We can now define our upper bound function UB_0; we defer the proof of validity of the upper bound to the online appendix [1].

DEFINITION 5 (UPPER BOUND). *Let* (X, t_p) *be a lattice element,* D_{dirty} *a dirty dataset, and* $\mathfrak{M}$ *a set of modifications.*

$$\mathsf{UB}_0(\mathsf{X}, t_p) := \max_{\varphi \in \mathsf{CandCFD}(\mathsf{X}, t_p)} |\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})|. \quad \square$$

PROPOSITION 2. *The upper bound function* $\mathsf{UB}_0(\cdot)$ *satisfies the conditions of Proposition 1.* □

We also introduce another upper bound, based on $\mathsf{UB}_0(\cdot)$, with the difference that it also takes into account the attributes in ModVIO covered by explanations. More specifically, for a CFD φ, we define $\mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})$ as the set of attributes occurring in $\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})$. Furthermore, we let λ be a parameter such that $0 \leq \lambda \cdot |\mathcal{A}| < 1$, where $|\mathcal{A}|$ is the number of attributes in the relation R. We define $\mathsf{UB}_\lambda(\mathsf{X}, t_p)$ as the maximum value of

$$|\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})| + 1 - \lambda|\mathsf{X} \cup \mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})|,$$

where, as before, φ ranges over all CFDs in $\mathsf{CandCFD}(\mathsf{X}, t_p)$. The intuition behind the additional negative term, compared to $\mathsf{UB}_0(\cdot)$, is to prioritize explanations to more general CFDs (containing a smaller number of attributes) during the execution of XPLODE. The additional "+1" is to ensure that $\mathsf{score}(\varphi, \mathfrak{M})$ remains smaller than $\mathsf{UB}_\lambda(\mathsf{X}, t_p)$ for every of its candidate CFDs φ, since $\lambda \cdot |\mathcal{A}| \leq 1$. Hence, $\mathsf{UB}_\lambda(\cdot)$ only affects the priority among those CFDs explaining an identical number of modifications. The proof that $\mathsf{UB}_\lambda(\cdot)$ is loose anti-monotonic is deferred to the online appendix [1].

4.3 Discovering Multiple Explanations

As mentioned in the introduction, we have devised algorithm XPLODE to be the core component of an interactive cleaning system. While finding the best explanation given a set of modifications is the crucial step in this system, in practice one might desire the algorithm to return multiple CFDs. We briefly discuss how XPLODE can be altered to discover (a) the top-k explanations for the current set of modifications, and (b) a sequence of i modifications that incrementally explain the modifications. We believe a combination of these techniques covers most real-world scenario's.

Discovering Top-k Explanations. Turning XPLODE into a top-k algorithm requires limited changes to the pseudocode shown in Algorithm 1. On line 4, we now initialize $\varphi_{\max}$ as a list of length k. On line 7, the algorithm has to be changed such that all $\varphi \in \mathsf{CandCFD}(\mathsf{X}, t_p)$ with score $>$ max are processed, not just the highest-scoring φ. Finally, on line 9, the identified φ is *added* to (the list) $\varphi_{\max}$, and if more than k CFDs are present in $\varphi_{\max}$, the lowest-scoring one is removed. The value of max is subsequently set to the *lowest* score of those CFDs in $\varphi_{\max}$.

Incrementally Explaining Modifications. We next discuss how to discover CFDs that incrementally explain observed modifications. In other words, the best explanation is first discovered, and then the search is continued in order to find the best explanation for the modifications that *have not yet been explained*. To do this efficiently, we first make a change to the lattice elements: instead of associating a *score* with each CFD, we attach the *set of modifications* the CFD explains. This allows us to efficiently recompute the score of a CFD after removing already-explained modifications.

The main change to the algorithm is that we introduce a list *backup* to store generated lattice elements, *in addition* to the priority queue. After line 15 in the algorithm, each generated lattice element (Y, s_p) is inserted into *backup* if $\mathsf{UB}(\mathsf{Y}, s_p) > 0$, i.e., if a CFD in the element or its children can explain some modification. When the regular XPLODE algorithm finishes, because the best explanation $\varphi_{\max}$ is found, we now remove all modifications explained by $\varphi_{\max}$ from $\mathfrak{M}$. Subsequently, the list *backup* is examined, and the scores of its elements are updated. All elements (Y, s_p) with $\mathsf{UB}(\mathsf{Y}, s_p) = 0$ are removed, and the others are used to reinitialize the priority queue Φ. The algorithm then repeats, until all modifications are explained.

4.4 Implementation Details

We conclude this section by elaborating on how XPLODE checks the support and confidence thresholds and how global explanations are filtered out. These all crucially rely on so-called equivalence partitions, commonly used in (C)FD discovery algorithms.

Equivalence Partitions. These partitions, introduced in [12] and used in CTANE [16] for CFD discovery (and in its predecessor TANE [22] for FD discovery), are compact representations of sets of tuples that *agree* on sets of attributes.

More specifically, given (X, t_p), where X is a set of attributes and t_p is a pattern tuple over X, we say that two tuples s and t in D_{rep} are *equivalent relative to* (X, t_p) if $s[\mathsf{X}] = t[\mathsf{X}] \asymp t_p$. For a tuple $s \in D_{rep}$, $[s]_{(\mathsf{X},t_p)}$ denotes the *equivalence class* consisting of the tids of all tuples $t \in D_{rep}$ that are equivalent with s relative to (X, t_p). The *(equivalence) partition of* (X, t_p), denoted by $\Pi(\mathsf{X}, t_p)$, is the collection of $[s]_{(\mathsf{X},t_p)}$ for $s \in D_{rep}$. The *size* of $\Pi(\mathsf{X}, t_p)$, denoted by $|\Pi(\mathsf{X}, t_p)|$, is the number of equivalence classes in $\Pi(\mathsf{X}, t_p)$. We use $\|\Pi(\mathsf{X}, t_p)\|$ to denote the number of tids in $\Pi(\mathsf{X}, t_p)$. For instance, in the running example we have that $\Pi(\{\mathsf{CC}, \mathsf{CT}\}, (44, \_)) = \{\{5, 6\}, \{7\}\}$ with size $|\Pi(\{\mathsf{CC}, \mathsf{CT}\}, (44, \_))| = 2$ and $\|\Pi(\{\mathsf{CC}, \mathsf{CT}\}, (44, \_))\| = 3$.

One of the key uses of equivalence partitions in discovery algorithms is to check the validity of CFDs [16]. It is easily verified that $D_{rep} \models (\mathsf{X} \to \mathsf{A}, t_p)$ if and only if $|\Pi(\mathsf{X}, t_p[\mathsf{X}])| = |\Pi(\mathsf{X} \cup \{\mathsf{A}\}, t_p)|$. That is, the number of equivalence classes relative to $(\mathsf{X}, t_p[\mathsf{X}])$ remains the same when adding $(\mathsf{A}, t_p[\mathsf{A}])$. Yet another way of phrasing this is as follows. For an equivalence class $\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])$, we let $\mathsf{Refine}(\mathsf{eq}, (\mathsf{A}, t_p[\mathsf{A}]))$ be the set of equivalence classes in $\Pi(\mathsf{X} \cup \{\mathsf{A}\}, t_p)$ that subsume eq. Then, $D_{rep} \models \varphi$ if and only if $|\mathsf{Refine}(\mathsf{eq}, (\mathsf{A}, t_p[\mathsf{A}]))| = 1$ for every $\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])$. We use this formulation below.

Equivalence partitions can also be used to check support and confidence thresholds and for checking whether or not a CFD is a global explanation, as will be explained below. For this reason, we compute equivalence partitions during XPLODE's lattice traversal. More precisely, just as in CTANE, we start by computing equivalence partitions for attribute/value pairs (A, a) with $a \in \mathsf{dom}(\mathsf{A}) \cup \{\_\}$ with support at least δ. Then, when children are generated, the equivalence partition for (X, t_p) is obtained by intersecting the partitions of two parents of (X, t_p). Here, intersecting means intersecting every pair of equivalence classes from the two partitions. We implement this intersection as in TANE by means of a linear time algorithm based on lookup tables [22]. Only partitions of high support are retained. Since support is anti-monotonic, neither elements with insufficient support nor their children need to be considered.

Checking for (ε, δ)-CFDs. Consider a CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$. Since the corresponding lattice element $(\mathsf{X} \cup \{\mathsf{A}\}, t_p)$ was added to the priority queue, it must have sufficient support. Due to the anti-monotonicity of support, the same is true for all its subsets, and hence $\mathsf{supp}(\varphi, D_{rep}) \geq \delta$. Checking whether $\mathsf{conf}(\varphi, D_{rep}) \geq 1 - \varepsilon$ can be done in terms of equivalence partitions along the same lines as the *error* of FDs is computed in TANE [22]. It is now easily verified (as in [22]) that $\mathsf{conf}(\varphi, D_{rep})$ is equal to

$$\frac{1}{\|\Pi(\mathsf{X}, t_p[\mathsf{X}])\|} \left(\sum_{\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])} |\mathsf{eq}| - \max_{\mathsf{eq}' \in \mathsf{Refine}(\mathsf{eq}, (\mathsf{A}, a))} |\mathsf{eq}'| \right).$$

Indeed, intuitively, this expression tells that in order to resolve violating tuples with tids in eq by deleting a minimal number of tuples (as required by the definition of confidence), all tuples have to be removed that do belong to classes in $\mathsf{Refine}(\mathsf{eq}, (\mathsf{A}, a))$, except for one class of maximal size. We note that $\mathsf{conf}(\varphi, D_{rep})$ is computed when element $(\mathsf{X} \cup \{\mathsf{A}\}, t_p)$ is considered. By contrast to CTANE, element $(\mathsf{X}, t_p[\mathsf{X}])$, needed to compute $\mathsf{conf}(\varphi, D_{rep})$, may not have been visited yet due to the way the lattice is traversed. In this case, we compute $\Pi(\mathsf{X}, t_p[\mathsf{X}])$ on-the-fly from the equivalence classes $\Pi(\mathsf{B}, b)$ for $\mathsf{B} \in \mathsf{X}$ and $b = t_p[\mathsf{B}]$.

Checking for global explanations. In addition to satisfying support and confidence thresholds, for a CFD to be a global explanation, three more conditions (as stated in Definition 3) need to be verified. We next show how these checks can be done by using equivalence partitions. As before, let $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$. We start with condition (3), as it is the easiest one. Recall that this condition states that we wish to discover CFDs φ that are not violated on repaired tuples. In other words, $\mathsf{VIO}(\varphi, \sigma_{\mathfrak{M}}(D_{rep}))$ should be empty. As already mentioned, checking for violations corresponds to finding equivalence classes $\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])$ such that $|\mathsf{Refine}(\mathsf{eq}, (\mathsf{A}, t_p[\mathsf{A}]))| > 1$. Since condition (3) only applies to tuples in $\sigma_{\mathfrak{M}}(D_{rep})$, it suffices to check whether there is an equivalence class $\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])$ for which there are tids in $\sigma_{\mathfrak{M}}(D_{rep})$ that belong to two different classes in $\mathsf{Refine}(\mathsf{eq}, (\mathsf{A}, t_p[\mathsf{A}]))$. Such a check can be easily integrated during the confidence computation of φ.

Conditions (1) and (2) in Definition 3 are a bit more challenging, as they require computations over D_{dirty}, whilst we only have equivalence partitions over D_{rep}. Suppose for the moment that we also have equivalence partitions over D_{dirty} at our disposal. We denote these by $\Pi_d(\mathsf{X}, t_p[\mathsf{X}])$ (with subscript "d" for dirty). Recall that condition (2) requires that $\mathsf{VIO}(\varphi, D_{dirty}) \cap \sigma^{\mathsf{tid}}_{\mathfrak{M}}(D_{dirty})$ is not empty. Given equivalence classes over D_{dirty}, this condition can be checked along the same lines as done for condition (3). Indeed, we compute $\mathsf{conf}(\varphi, D_{dirty})$ and along the way we check for violations involving tids in $\sigma_{\mathfrak{M}}(D_{dirty})$, just as before. As a positive side-effect, condition (1) requires comparing $\mathsf{conf}(\varphi, D_{dirty})$ and $\mathsf{conf}(\varphi, D_{rep})$, both of which are now already computed.

Pulling back the equivalence partitions. It remains to explain how equivalence partitions $\Pi_d(\mathsf{Y}, s_p)$ are computed. Instead of recomputing them from scratch on D_{dirty}, we "pull them back" from $\Pi(\mathsf{Y}, s_p)$, the partition of the element in D_{rep}. A description and pseudocode of this technique is available in the online appendix [1].

5. APPROXIMATING THE SCORE

As already observed in Section 3.3, the computation of $\mathsf{score}(\varphi, \mathfrak{M})$, as defined in Definition 4, is quite expensive. Indeed, it requires the traversal of a power set lattice whose elements consist of *all subsets* of the modifications in $\mathfrak{M}$. In this section we propose an approximate scoring function, denoted by $\mathsf{UC\text{-}score}$[2], which is easy to compute. Moreover, experiments show that $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M})$ is a good approximation of $\mathsf{score}(\varphi, \mathfrak{M})$. We show that $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leq \mathsf{score}(\varphi, \mathfrak{M})$ and hence, $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leq \mathsf{UB}_0(\mathsf{X}, t_p)$ (or $\mathsf{UB}_\lambda(\mathsf{X}, t_p)$). This implies that XPLODE can also be used for computing the global explanation with maximal $\mathsf{UC\text{-}score}$.

5.1 Rationale Behind UC-score

We first observe that for constant CFDs φ, different violations are *independent* of each other (after all, constant CFDs concern violations consisting of single tuples only). Furthermore, let us call a set $M \subseteq \mathfrak{M}$ *valid* if every modification in M refers to a unique tuple. In other words, no two modifications in a valid set relate to the same tuple. Then, for constant CFDs and valid sets M, if each *single* modification $\mathfrak{m} \in M$ is locally explained by φ, then also the *entire set* M is locally explained by φ. No exhaustive enumeration of subsets of M is thus needed to check for local explainability. To obtain the best possible approximation of $\mathsf{score}(\varphi, \mathfrak{M})$ in this way, it thus suffices to count the number of tids that occur in a modification in $\mathfrak{M}$ that is locally explained by φ.

DEFINITION 6 ($\mathsf{UC\text{-}score}(\cdot)$, CONST. CFD). *Let φ be a constant CFD and $\mathfrak{M}$ a set of modifications. We define*

$$\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) := \max\{|M| \mid M \subseteq \mathfrak{M} \text{ is valid and each } \mathfrak{m} \in M \text{ is locally explained by } \varphi\}. \quad \square$$

As observed earlier, the $\mathsf{UC\text{-}score}(\cdot)$ is equal to the size of the largest valid set M that is locally explained.

The situation for variable CFDs is quite different, however, due to *dependencies* between violations.

EXAMPLE 6. *Consider modification $\mathfrak{m}_3 = (8, \mathsf{CC}, 44, 01)$ from the running example. A variable CFD that locally explains this modification is $\varphi = (\mathsf{CC} \to \mathsf{PN}, (\_, \_))$. Now, assume a different modification, $\mathfrak{m}_4 = (3, \mathsf{PN}, 2222222, 1111111)$. By itself, this modification is also explained by φ. When applying both modifications, however, tuples t_3, t_8 have $\mathsf{CC} = 01$, but a different PN, violating the CFD φ.* $\diamondsuit$

To define an efficient, yet useful scoring function for variable CFDs, we will treat a variable CFD $\varphi = (\mathsf{X} \to \mathsf{A}, (t_p, \_))$ as a *union* of a finite number of *constant* CFDs, say $\Sigma = \{\varphi_1, \cdots, \varphi_m\}$[3]. Moreover, when we allow unions of constant CFDs to serve as the constraint language for global explanations, they inherit the nice properties of single constant CFDs, with some restrictions.

DEFINITION 7 ($\mathsf{UC\text{-}score}(\cdot)$, UNION OF CONST. CFDs). *Let φ be a CFD, $\Sigma_\varphi = \{\varphi_1, \cdots, \varphi_m\}$ a union of constant CFDs, and $\mathfrak{M}$ a set of modifications. We define*

$$\mathsf{UC\text{-}score}(\Sigma_\varphi, \mathfrak{M}) := \max\{|M| \mid M \subseteq \mathfrak{M} \text{ is } \Sigma\text{-valid and each } \mathfrak{m} \in M \text{ is locally explained by } \varphi\}. \quad \square$$

It again holds that $\mathsf{UC\text{-}score}(\Sigma, \mathfrak{M})$ is the size of the largest Σ-valid set of modifications that is locally explained by Σ, and can be computed as the number of tids that occur in a modification in $\mathfrak{M}$ that is locally explained by Σ. This property is crucial for the efficient computation of $\mathsf{UC\text{-}score}(\Sigma, \mathfrak{M})$. We explain the notion of Σ-valid set in the next section.

Consider now a variable CFD $\varphi = (\mathsf{X} \to \mathsf{A}, (t_p, \_))$. We convert φ into a union Σ_φ of constant CFDs as follows: for each equivalence class $\mathsf{eq} \in \Pi_d(\mathsf{X}, t_p)$, we denote by c_{eq} the projection on the X-attributes of a tuple in eq (more precisely, a tuple with a tid in eq). Furthermore, we let a_{eq} be the most frequent A-value in all tuples in eq. We then define $\varphi_{\mathsf{eq}} = (\mathsf{X} \to \mathsf{A}, (c_{\mathsf{eq}}, a_{\mathsf{eq}}))$ and represent φ as the union $\Sigma_\varphi = \{\varphi_{\mathsf{eq}} \mid \mathsf{eq} \in \Pi_d(\mathsf{X}, t_p)\}$ of constant CFDs. Intuitively, the most frequent A-value in each equivalence class is expected to reflect the correct value in that equivalence class. More importantly, recall that the confidence of φ (see Section 4.4) is computed by "removing tuples that do belong to classes in $\mathsf{Refine}(\mathsf{eq}, (\mathsf{A}, a))$, except for those tuples in the class of maximal size". Thus, the most frequent A-value directly relates to the confidence of the CFD. In conclusion, given a variable CFD φ, we define

DEFINITION 8 ($\mathsf{UC\text{-}score}(\cdot)$ OF A CFD). *Let φ be any CFD, $\Sigma = \{\varphi_1, \cdots, \varphi_m\}$ a union of constant CFDs, and $\mathfrak{M}$ a set of modifications. We define*

$$\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) := \mathsf{UC\text{-}score}(\Sigma_\varphi, \mathfrak{M}). \quad \square$$

In the remainder of this section we describe the crucial property underlying the definition of $\mathsf{UC\text{-}score}$, show that $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M})$ is smaller than or equal to $\mathsf{score}(\varphi, \mathfrak{M})$, and verify that $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M})$ is easy to compute. The proofs of these properties can be found in the online appendix [1].

5.2 Properties of UC-score

Consider a variable CFD $\varphi = (\mathsf{X} \to \mathsf{A}, (t_p, \_))$ and let $\Sigma_\varphi = \{\varphi_{\mathsf{eq}} \mid \mathsf{eq} \in \Pi_d(\mathsf{X}, t_p)\}$ be the set of constant CFDs obtained from φ [4]. We call a set $M \subseteq \mathfrak{M}$, Σ_φ-*valid* if it is valid and, in addition, for all tuples $t \in \sigma_M(D_{dirty} \oplus M)$ there either exists a constant CFD $\varphi_{\mathsf{eq}} = (\mathsf{X} \to \mathsf{A}, (c_{\mathsf{eq}}, a_{\mathsf{eq}})) \in \Sigma_\varphi$ such that $t[\mathsf{X}] = c_{\mathsf{eq}}$, or $t[\mathsf{X}] \not\asymp t_p$. Intuitively, a set M of

[2]The "UC" in UC-score refers to Unions of Constant CFDs.

[3]We represent a union of CFDs as a set of CFDs.

[4]To uniformly treat variable and constant CFDs, for a constant CFD φ we let Σ_φ be the singleton CFD $\{\varphi\}$.

modifications is Σ_φ-valid when in $\sigma_M(D_{dirty} \oplus M)$, either the violations of φ on D_{dirty} are repaired in accordance with the constant CFDs in Σ_φ, or these violations are repaired by invalidating the constants in the pattern tuple t_p of φ. By focusing on a set Σ_φ of constant CFDs, and Σ_φ-valid sets of modifications, we can indeed efficiently approximate score:

PROPOSITION 3. For any Σ_φ-valid set of modifications $M \subseteq \mathfrak{M}$, M is locally explained by Σ_φ if and only if each $\mathtt{m} \in M$ is locally explained by Σ_φ. Furthermore, $\mathsf{UC\text{-}score}(\Sigma_\varphi, M)$ is equal to the number of tids that occur in a modification in $\mathfrak{M}$ that is locally explained by Σ_φ. □

In other words, computing the score relative to M does not require an exhaustive exploration of all subsets of M, in contrast to the computation of $\mathsf{score}(\varphi, \mathfrak{M})$ defined in Section 3.3. An important property is that:

PROPOSITION 4. For every global explanation φ and set $\mathfrak{M}$ of modifications, $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leq \mathsf{score}(\varphi, \mathfrak{M})$.

PROOF. (sketch) We show that if Σ_φ locally explains a Σ_φ-valid set M, then φ also locally explains M. Thus, $\mathsf{UC\text{-}score}(\Sigma_\varphi, M) \leq \mathsf{score}(\varphi, \mathfrak{M})$ for every Σ_φ-valid M. □

Consequently, $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leq \mathsf{score}(\varphi, \mathfrak{M}) \leq \mathsf{UB}_0(\mathsf{X}, t_p)$ (and $\mathsf{UB}_\lambda(\mathsf{X}, t_p)$) when φ is of the form $(\mathsf{X} \backslash \mathsf{A} \to \mathsf{A}, t_p)$; hence XPLODE finds the global explanation of highest UC-score.

5.3 Computation of UC-score

We conclude by explaining how $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M})$ can be efficiently computed. Proposition 3 tells that it suffices to count the number of tids that occur in a modification in $\mathfrak{M}$ that is locally explained by Σ_φ. This is checked as follows:

PROPOSITION 5. Let Σ_φ be the set of constant CFDs of CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$. Let $\mathtt{m} = (\mathsf{tid}, \mathsf{B}, v_d, v_c) \in M$, with $M \subseteq \mathfrak{M}$ a Σ-valid set of modifications, $s = D_{dirty}[\mathsf{tid}]$ and $t = (D_{dirty} \oplus \mathtt{m})[\mathsf{tid}]$. Then Σ_φ locally explains $\mathtt{m}$ if and only if there exists a constant CFD $\varphi_{\mathsf{eq}} = (\mathsf{X} \to \mathsf{A}, (c_{\mathsf{eq}}, a_{\mathsf{eq}})) \in \Sigma_\varphi$ such that $s[\mathsf{X}] = c_{\mathsf{eq}}$ and $s[\mathsf{A}] \neq a_{\mathsf{eq}}$ (s *violates* φ_{eq}), and:

1. either $t[\mathsf{A}] = a_{\mathsf{eq}}$ (t *satisfies* φ_{eq}); or
2. there exists another $\varphi_{\mathsf{eq}'} \in \Sigma_\varphi$ such that $t[\mathsf{X}] = c_{\mathsf{eq}'}$ and $t[\mathsf{A}] = a_{\mathsf{eq}'}$ (t *satisfies some other CFD in* Σ_φ); or
3. $t[\mathsf{X}] \not\asymp t_p$ (*(φ no longer applies to t).* □

Of course, for constant CFDs φ, $\Sigma_\varphi = \{\varphi\}$ and hence only cases (1) and (3) in the Proposition apply. Although Definition 3 also requires checking whether $\mathsf{conf}(\Sigma_\varphi, D_{dirty}) < \mathsf{conf}(\Sigma_\varphi, D_{dirty} \oplus \mathtt{m})$, as part of the proof of Proposition 5 we show that this is implied by the conditions in its statement.

The pseudo-code for computing UC-score, shown in Algorithm 2, is based on Propositions 3 and 5. We first convert the CFD φ into its set Σ_φ, using function CONVERTCFD, as previously explained. Then, $\mathsf{VIO}(\Sigma_\varphi, D_{dirty})$ is computed. By Proposition 5 it suffices to only consider $\mathtt{m} \in \mathfrak{M}$ that relate to tids in $\mathsf{VIO}(\Sigma_\varphi, D_{dirty})$. We partition modifications in $\mathfrak{M}$ according to their tid. Let $\mathfrak{M}[\mathsf{tid}]$ be the set of modifications in $\mathfrak{M}$ that relate to tid. Since every modification occurs in exactly one attribute, we can further partition $\mathfrak{M}[\mathsf{tid}]$ into $\mathfrak{M}[\mathsf{tid}, \mathsf{X}]$ and $\mathfrak{M}[\mathsf{tid}, \mathsf{A}]$, consisting of modifications on attributes in X and A, respectively. For modifications $\mathtt{m} = (\mathsf{tid}, \mathsf{B}, v_d, v_c)$ in $\mathfrak{M}[\mathsf{tid}, \mathsf{X}]$, we increment the score on two occasions: (i) on line 7, if $t_p[\mathsf{B}]$ is a constant for the attribute B in which a change happens, then clearly the change makes φ inapplicable to the tuple in question (condition 3 in Proposition 5); and (ii) on lines 9 and 10, if the change results in the tuple satisfying some constant CFD $\varphi_{\mathsf{eq}'} \in \Sigma_\varphi$ (condition 2 in Proposition 5). Finally, on lines 13–14, we perform a similar computation for modifications in $\mathfrak{M}[\mathsf{tid}, \mathsf{A}]$: We determine the constant CFD φ_{eq} that was violated in the tuple s in D_{dirty}, and increment the score if attribute A was modified such that tuple t satisfies φ_{eq} (condition 1 in Proposition 5). As soon as a modification is explained for a given tid t, it is counted, and the algorithm proceeds to the next tid.

Algorithm 2 Computing the UC-score of a CFD φ.

```
1: procedure UC-SCORE(D_dirty, D_rep, 𝔐, φ : (X → A, t_p))
2:   Σ_φ ← CONVERTCFD(φ)
3:   ucscore ← 0
4:   for all tid ∈ VIO(Σ_φ, D_dirty) do
5:     𝔐[tid] ← {m ∈ 𝔐 | m relates to tid}
6:     for all m = (tid, B, v_d, v_c) ∈ 𝔐[tid, X] do
7:       if t_p[B] ≠ '_' then
8:         Increment ucscore and go to next tid
9:       else if ∃φ_eq', t = D_rep[tid], t[X] = c_eq' then
10:        if t[A] = a_eq' then
11:          Increment ucscore and go to next tid
12:    for all m ∈ 𝔐[tid, A] do
13:      if ∃φ_eq, s = D_dirty[tid], s[X] = c_eq then
14:        if t[A] = a_eq for t = D_rep[tid] then
15:          Increment ucscore and go to next tid
16:  return ucscore
```

Table 2: Statistics of the used datasets.

Dataset	#Tuples	#Attributes	%MinSupp
Abalone	8354	9	10%
Adult	97684	11	1%
Soccer	200000	10	10%
SP500	245148	7	1%

6. EXPERIMENTS

We experimentally validate our repair explanation method. Experiments illustrating the benefit of the optimizations discussed in Section 4, are postponed to the online appendix [1].

6.1 Experimental Setup

Hardware. All our experiments were performed on an Intel Core i7 Processor (2.3GHZ) with 16GB of memory running OS X. All algorithms are implemented in C++ and run entirely in main memory. All code and data is available on the CodeOcean platform for reproducible research[5].

Datasets. We use four datasets: **Abalone** and **Adult** are small datasets from the UCI Repository [27]; **Soccer** is a synthetic dataset about soccer players and their teams[6], and finally, **SP500** is a real-world dataset about stock trading from [10]. To ensure that CFD violations can occur, we duplicate every tuple in these datasets. On the Adult dataset we only use constants CFDs, since mining general CFDs on this dataset is too time-consuming, because Adult has a higher number of attributes and many frequent constant patterns. Statistics of the data are shown in Table 2.

[5] https://bit.ly/2MYzjcH

[6] http://www.db.unibas.it/projects/bart/

Error Generation. We make use of the BART tool [2] for introducing violations in the datasets. BART takes a dataset and a set of data quality rules as input, and inserts a predefined percentage of violations into the data. The used percentages are reported in the **%Error** column in Table 4. To get the required quality rules, we used our implementation of CTane [16] to discover 100% confident CFDs on the datasets, using the minimum support percentages shown in Table 2. As a note aside, support threshold δ is then simply $\frac{1}{100}$(**#Tuples**×**%MinSupp**). These thresholds were set empirically, low enough to ensure that a reasonable number of 100% confident CFDs were found on the datasets, i.e., at least 50; yet high enough for the runtime to remain practical for experimenting. Combining the clean and dirty datasets, we generate *partial* repairs by starting from the dirty dataset, and replacing a subset of the dirty tuples with their clean variants. For each dataset, we obtain 3 different dirty datasets by using 3 different CFDs discovered on the clean data, denoted by CFD 1, CFD 2, and CFD 3. When considering a dirty dataset corresponding to a CFD i, then CFD i is the *target* CFD. In other words, it is the CFD we want to discover by repairing the corresponding dirty dataset. Obviously, CFD i is different for each dataset.

Falcon. We compare with Falcon [21], a system which discovers a SQL Update Statement (equivalent to a constant CFD) that explains a *single* modification. Since such a modification does not contain sufficient information to reliably discover the underlying CFD, Falcon employs *user interaction* to narrow down the search space. Feedback is received in the form of a user asserting the (in)validity of a given CFD. Using the fact that all generalizations of an invalid CFD are also invalid, and all specializations of a valid CFD are also valid, an efficient binary search algorithm limits the amount of user feedback required.

6.2 Experimental Results

Usefulness of Explaining from Repairs. Before evaluating the performance of our method for discovering CFDs that explain repairs, we first show that user-supplied modifications are indeed a good instrument to guide CFD discovery towards a CFD that is useful for repairing. We perform CFD discovery on the dirty data, and rank the discovered approximate CFDs by confidence and rule length, and compare this to a ranking using $\mathsf{UC\text{-}score}(\cdot)$ on partially cleaned data, with 2 and 5 modifications, respectively.

In Table 3, we show the position of the target CFD in a list of CFDs ranked according to the different criteria (position 1 denotes the top of the ranking). The results clearly show that, when using confidence or rule size, the target is typically quite deep in the ranking. It is infeasible for a user to manually validate all of these CFDs one by one. This illustrates the need for integrating user feedback into the ranking function. Indeed, using information from modifications consistently quickly brings the target to the front, ratifying our approach. Repairing a handful of tuples automatically invalidates many CFDs, saving much user effort.

Explaining Full Repairs. To illustrate the efficacy of our method, we first perform explanation discovery on *full* repairs, i.e., using the fully clean data as D_{rep}. We are thus only interested in CFDs that are not violated on D_{rep}, and set $\varepsilon = 0$. The minimum support threshold is set according to the percentages from Table 2. These thresholds ensure that the target CFD is among the (ε, δ)-CFDs that are the candidates for global explanations. In this experiment, we first want to get an idea about the total number of global explanations. Therefore, we use the post-processing method: Run CTane to find (ε, δ)-CFDs; then filter out all the global explanations. We find that the number of $(0, \delta)$-CFDs that are global explanations is typically too large for manual inspection, ranging from around 40 on the Adult dataset up to 400 on the Soccer dataset. On partial repairs (when $\varepsilon > 0$), this number will only increase since the number of (ε, δ)-CFDs will increase considerably, up to 1500 on Abalone for $\varepsilon = 0.1$. Secondly, we want to validate that the target CFD is indeed the highest scoring (for $\mathsf{UC\text{-}score}(\cdot)$) global explanation when considering full repairs. Indeed, on all datasets the target CFD was discovered. It shows that the $\mathsf{UC\text{-}score}(\cdot)$ is a good measure for repair explanations.

Scoring Function. We have also evaluated whether the scoring function $\mathsf{UC\text{-}score}(\cdot)$ is indeed a suitable surrogate for $\mathsf{score}(\cdot)$, the exact scoring function. Since $\mathsf{score}(\cdot)$ requires an expensive computation, which we implemented using a brute-force approach, we performed the experiment only on the smallest dataset, Abalone. We computed both the ranking difference and the absolute difference between both scoring functions. Due to space limitations, we postpone the results to the online appendix [1]. We learn that the top positions in the ranking are unaffected by the choice of scoring function. Moreover, the absolute error remains very small throughout the entire ranking. Hence, $\mathsf{UC\text{-}score}(\cdot)$ offers a good approximation for $\mathsf{score}(\cdot)$, and is unlikely to change which global explanation is returned by XPlode, compared to $\mathsf{score}(\cdot)$. We use $\mathsf{UC\text{-}score}(\cdot)$ in all remaining experiments.

Explaining Partial Repairs. Our first experiment showed that the target CFD, the one used to dirty the data, can be recovered from a full repair. We now want to answer the question, "how many modifications are needed for a partial repair to recover the target CFD?". We created partial repairs using an increasing number of modifications, and report the number and percentage of modifications needed until XPlode returns the target CFD. On each dataset, we use the error percentage of the data as ε in the confidence threshold $1 - \varepsilon$; the support threshold δ is derived from the difference of (i) the minimum support percentage used to mine the CFDs and (ii) the error percentage of the data. For instance, on Abalone with 1% errors, we pick $\delta = (10\% - 1\%) \times 8354$ and $\varepsilon = 0.01$. As before, these guarantee that the target CFD is among the (ε, δ)-CFDs that are explored. Furthermore, we use $\mathsf{UB}_\lambda(\cdot)$ as upper bound function in XPlode. Experiments (not reported) show that this loose anti-monotonic function guides XPlode more quickly towards the desired CFD than $\mathsf{UB}_0(\cdot)$, since the λ penalty assigns a lower priority to lattice elements containing irrelevant attributes, i.e., attributes that do not occur in AttVIO.

The results for this experiment on all four datasets and CFD 1, CFD 2, CFD 3, are shown in Table 4. Here, the columns **%Error** and **#Error** contain, respectively, the percentage and absolute number of violations inserted by BART. The columns **%M(i)** and **#M(i)** display the percentage and number of modifications required before the target CFD is returned, for CFD i. We see that the percentage of modifications required to find the target CFD is typically low, and does not change much when the dirtiness of the data increases. This implies that our method has a greater benefit when cleaning dirtier data: if the target CFD can be

Table 3: Position of target CFD among all approximate CFDs according to various ranking criteria.

Dataset	CFD	Length (Asc)	Conf (Asc)	Conf (Desc)	UC-score$_2(\cdot)$	UC-score$_5(\cdot)$
Abalone	1	907	249	4617	18	2
	2	1825	1500	3703	12	3
	3	2492	305	4565	244	3
Adult	1	153006	143303	62202	256	10
	2	31948	141376	64517	15	1
	3	10064	191665	27548	2	1
Soccer	1	3896	806	11953	49	3
	2	1505	1424	11046	3	3
	3	1232	1385	12329	14	9
SP500	1	150	126	258	4	1
	2	171	127	248	42	2
	3	166	127	255	46	2

Table 4: Number and percentage of modifications required to retrieve the target CFD, for 3 different CFDs.

Dataset	%Error	#Error	%M(1)	#M(1)	%M(2)	#M(2)	%M(3)	#M(3)
Abalone	0.1	8	100%	8	50%	4	100%	8
	1	83	10%	8	5%	4	14%	12
	10	835	1%	8	1%	8	2%	18
Adult	0.1	97	20%	19	6%	6	30%	29
	0.5	488	5%	24	2%	12	0.6%	3
	1	976	2%	24	0.4%	4	5%	48
Soccer	0.1	200	9%	17	11%	22	1%	2
	1	2000	0.3%	7	1%	22	0.1%	2
	10	20000	0.2%	30	0.1%	20	0.1%	25
SP500	0.1	245	3%	7	3%	7	0.8%	2
	0.5	1225	0.5%	7	0.5%	7	0.1%	2
	1	2451	0.1%	3	0.1%	3	0.1%	3

found by manually cleaning 1% of the violations, then 99% of the violations may be cleaned automatically. Moreover, the number of attributes has a higher impact on the number of required modifications than the number of tuples.

Comparison with Falcon. We next compare XPlode and Falcon [21], for the case where the target CFD is a constant CFD[7] The experiment was done on the Soccer dataset, using 3 constant CFDs. Only modifications that relate to the consequent of the CFDs were considered, as Falcon supports only these. Falcon was able to recover each of the target CFDs using a single modification and 2 user questions, taking between 1 and 4 seconds. In comparison, XPlode recovers the target CFD using 3 to 5 modifications, with a runtime around 4 seconds each time. Since the runtimes were obtained on different hardware, it is hard to compare efficiency. Nevertheless, the experiment suggests that XPlode, although not specialized only towards constant CFDs, is comparable to Falcon.

We remark that this changes in the case of variable CFDs. Indeed, in this case Falcon finds a constant CFD for every constant pattern relating to the variable CFD. For each such CFD, a single modification and some user questions would be required. We consider again the Soccer dataset, this time using 3 variable CFDs. XPlode can recover the single target CFD using, on average, 12 modifications. However, in order to capture all the errors in these datasets using constant CFDs, on average 55 constant CFDs are needed. This implies that Falcon would require this amount of user modifications, and outputs a large number of CFDs as well. A similar scenario occurs when considering modifications on attributes in the antecedent of CFDs, since Falcon assumes that modifications pertain only to the *consequent* of a rule.

[7]Due to IP restrictions, we were unable to obtain the code and perform an in-depth comparison. P. Papotti and E. Veltri, co-authors of [21], kindly performed an experiment.

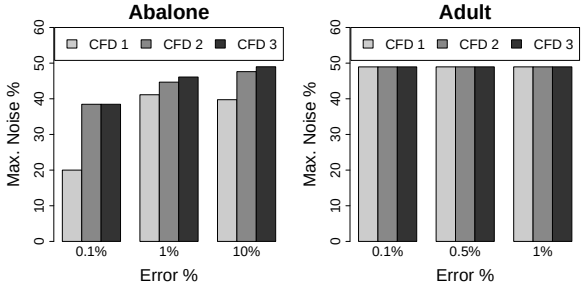

Figure 1: Noise-robustness of XPlode.

Robustness to Noise. So far, we have run experiments using only modifications that belong to a single CFD, and as such, the returned CFD explains all the given modifications. However, when a user is manually cleaning data, without knowing the target CFD, corrections may be made in positions unrelated to the target CFD. We now verify whether XPlode is robust to such "noise".

We consider again a full repair, and add random modifications throughout the data, not connected to the target CFD. In Figure 1, we report how many random modifications can be added without distorting the output of XPlode, up to 50% of the total number of modifications, i.e., as many random modifications as correct modifications. The results for SP500 and Soccer, available in the online appendix [1], are similar to Abalone.

The results show that our method is very robust to random noise, especially as more modifications are considered. On the Adult dataset, where we only consider constant CFDs, noise seems to have no impact. This make sense: as variable CFDs can capture a larger variety of errors, they are also more likely to accidentally explain a random modification, whereas constant CFDs are unable to connect the random modifications to each other.

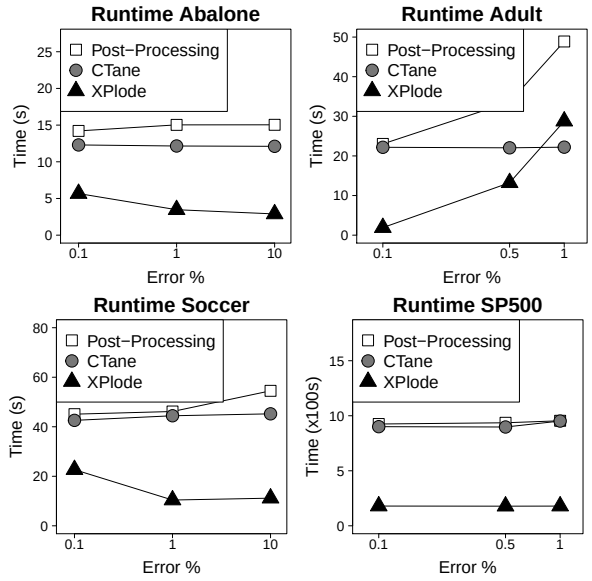

Figure 2: Runtime performance of XPlode, compared to post-processing and CTane.

Runtime Performance. Finally, we evaluate the runtime performance of XPlode on full repairs. We compare XPlode with two benchmarks: firstly, discovering all global explanations and then computing all the scores, and secondly, the discovery of *all* (ε, δ)-CFDs using the CTane algorithm. Figure 2 shows the runtimes (in seconds) averaged over 3 independent runs with different CFDs. Algorithm XPlode clearly outperforms post-processing in every case, and is typically faster than a full CTane execution. Overall, results indicate that the error percentage has little impact on runtimes. The only exception is the Adult dataset: here, the runtime deteriorates with a higher percentage of errors. We suspect that, for constant CFDs, checking for global explanations accounts for more of the total runtime. With more modifications, computing equivalence partitions in D_{dirty} becomes harder, as the difference between D_{dirty} and D_{rep} increases. Moreover, Adult shows a greater increase in the number of global explanations as dirtiness increases.

7. RELATED WORK

Our work is situated in the area of constraint-based data quality (see [14, 23] for surveys) and is based on CFDs [15].

Underlying XPlode is a constraint discovery process. By contrast to traditional (C)FD discovery algorithms [8, 16, 22, 31, 29] (see also [31, 28, 14, 23] for overviews) that aim to find *all* constraints that approximately hold on the data, we aim to find those CFDs that *explain a partial repair*. Traditional methods do not consider our notions of explanation and scoring function. To find explanations, these algorithms have to be combined with a post-processing step, which is expensive both in time and user interaction. In XPlode, post-processing is avoided by carefully integrating the notion of explanations in the discovery process. The resulting on-demand algorithm is similar in spirit to the method presented in [20], where constants patterns for a *fixed* FD are found that best describe the data. Our search strategy allows for general *loose anti-monotonic* functions [6], such as our scoring function, instead of the *strictly anti-monotonic* functions of support and confidence used in [20]. In addition, we explore the *entire* space of CFDs instead of requiring the embedded FD to be fixed. We leverage equivalence partitions [12] as commonly used in constraint discovery [22, 16]. None of the traditional methods consider user interaction, which is *essential* for our method. Other discovery methods that leverage user interaction are [21, 35]: The Falcon system [21], described in Section 6.1, is most closely related. It finds *constant* CFDs based on a *single* modification and afterwards relies on a "user oracle" to (in)validate the proposed rules. The emphasis is on limiting calls to this oracle. By contrast, we *only* use information stemming from *multiple* modifications and have no need for a "black box" oracle. Our method is capable of handling *variable* CFDs, providing more flexibility and more compact explanations of modifications, since a variable CFD can represent a large number of constant CFDs. In addition, our method is on-demand. The UGuide system [35] aims to find a set of FDs such that their *violation set* overlaps maximally with the tuples holding *true* errors, whilst minimizing the number of violations that are not true errors. We aim to explain *repairs* rather than errors. UGuide asks users to either (in)validate FDs, or specify whether given cells or tuples are true errors or not. It is a best effort method given a budget on the number of user interactions. We treat both types of user interaction *uniformly* through modifications: by incorporating information on *how* errors should be repaired, we can *zoom in* directly to CFDs that are *useful for repairing* and not only for error detection. By contrast, UGuide initially uses *all* approximate FDs, which is costly and leads to users invalidating an excessive amount of spurious dependencies. Finally, we consider general CFDs rather than just FDs. Other forms of user interaction in data cleaning are considered in works such as [37, 36, 39]. In these works, the set of (valid) constraints (FDs, CFDs, or other) is assumed to be available already. User involvement is used to determine the right repairs relative to the constraints. Our work leverages user interaction in an earlier phase, to *find valid constraints for repairing*. Finally, we rely on users to make some initial repairs, after which existing data repairing algorithms [5, 13, 17, 18, 24, 26] can be used. Our approach is complementary to any of these repairing algorithms. In [7], a *description of errors* by means of a small set of conjunctive queries is discovered, without considering user interaction or repairing. Our repair explanations could also be of use to update constraints over evolving data [9, 4, 30].

8. CONCLUSION

We considered the problem of finding a CFD that best explains a partial repair. Our on-demand algorithm, called XPlode, shows great promise for finding such explanations. Underlying the efficacy of the method is a scoring function and its efficient approximation, that counts the number of modifications explained. A search space traversal based on loose anti-monotonic bounds on the scores, leads to an improvement in efficiency, compared to a post-processing approach to the problem. Experiments show high precision in discovering the correct explanation, using few modifications. Future work includes the investigation of alternative scoring functions, and extending XPlode towards discovering multiple explanations. Furthermore, the general concept of inferring constraints from a repair can be investigated in the context of other constraint formalisms [34, 38, 19, 11]. Suggesting modifications based on explanations also seems a natural extension of our framework.

9. REFERENCES

[1] Full version. https://bit.ly/2tU2zcF.

[2] 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*, 9(2):36–47, 2015.

[3] R. Bertens, J. Vreeken, and A. Siebes. Efficiently discovering unexpected pattern-co-occurrences. In *SDM*, pages 126–134, 2017.

[4] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552, 2013.

[5] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.

[6] F. Bonchi and C. Lucchese. Pushing tougher constraints in frequent pattern mining. In *PAKDD*, volume 5, pages 114–124, 2005.

[7] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *SIGMOD*, pages 445–456, 2014.

[8] F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

[9] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.

[10] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[11] X. Chu, I. F. Ilyas, P. Papotti, and Y. Ye. Ruleminer: Data quality rules discovery. In *ICDE*, pages 1222–1225, 2014.

[12] S. S. Cosmadakis, P. C. Kanellakis, and N. Spyratos. Partition semantics for relations. *Journal of Computer and System Sciences*, 33(2):203 – 233, 1986.

[13] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: A commodity data cleaning system. In *SIGMOD*, pages 541–552, 2013.

[14] W. Fan and F. Geerts. *Synthesis Lectures on Data Management: Foundations of data quality management*. Morgan & Claypool, 2012.

[15] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM TODS*, 33(2):6, 2008.

[16] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE TKDE*, 23(5):683–698, 2011.

[17] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. *Declarative data cleaning: Language, model, and algorithms*. PhD thesis, INRIA, 2001.

[18] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. That's all folks!: Llunatic goes open source. *PVLDB*, 7(13):1565–1568, 2014.

[19] L. Golab, H. Karloff, F. Korn, B. Saha, and D. Srivastava. Discovering conservation rules. *IEEE TKDE*, 26(6):1332–1348, 2014.

[20] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.

[21] 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*, pages 893–907, 2016.

[22] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

[23] I. F. Ilyas and X. Chu. *Trends in Cleaning Relational Data: Consistency and Deduplication*. Now Publishers Inc., 2015.

[24] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230, 2015.

[25] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129–149, 1995.

[26] S. Kolahi and L. V. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[27] M. Lichman. UCI machine learning repository, 2013.

[28] J. Liu, J. Li, C. Liu, and Y. Chen. Discover dependencies from data - A review. *IEEE Trans. Knowl. Data Eng.*, 24(2):251–264, 2012.

[29] P. Mandros, M. Boley, and J. Vreeken. Discovering reliable approximate functional dependencies. In *KDD*, pages 355–363, 2017.

[30] M. Mazuran, E. Quintarelli, L. Tanca, and S. Ugolini. Semi-automatic support for evolving functional dependencies. In *EDBT*, pages 293–304, 2016.

[31] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB*, 8(10):1082–1093, 2015.

[32] C. Pit-Claudel, Z. Mariet, R. Harding, and S. Madden. *Outlier Detection in Heterogeneous Datasets using Automatic Tuple Expansion*. Technical Report MIT-CSAIL-TR-2016-002, CSAIL, MIT, 32 Vassar Street, Cambridge MA 02139, February 2016.

[33] J. Rammelaere, F. Geerts, and B. Goethals. Cleaning data with forbidden itemsets. In *ICDE*, pages 897–908, 2017.

[34] S. Song and L. Chen. Discovering matching dependencies. In *CIKM*, pages 1421–1424, 2009.

[35] S. Thirumuruganathan, L. Berti-Equille, M. Ouzzani, J.-A. Quiane-Ruiz, and N. Tang. UGuide: User-guided discovery of FD-detectable errors. In *SIGMOD*, pages 1385–1397, 2017.

[36] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, 2014.

[37] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[38] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468, 2014.

[39] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.