Mining Patterns in Dirty Data for Detecting and Correcting Inconsistencies

Proefschrift

voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica
aan de Universiteit Antwerpen
te verdedigen door

Joeri Rammelaere

Promotor: prof. dr. Floris Geerts
prof. dr. Bart Goethals

Antwerpen, 2018

Mining Patterns in Dirty Data for Detecting and Correcting Inconsistencies
Nederlandse titel: *Patronen ontdekken in dirty data voor het detecteren en herstellen van inconsistenties*

Acknowledgements

Have no worries, reader. In just a few pages, the technical and scientific content commences. But before that, there is still time for some honourable mentions.

My journey towards a PhD started with a late evening e-mail to the supervisor of my master thesis, Bart Goethals. *Hi Bart, I want to do a PhD in data mining*. I received an uncharacteristically swift reply, and a few weeks later, I had a job interview for a position as teaching assistant. Afterwards, Bart told me he would inform me of the decision in three days. So, after five days, I reminded him. I did not get the job in the end (thanks Len), but I got something better (thanks Len!): a scholarship for a PhD on data cleaning, with the use of data mining, under supervision of Floris Geerts.

So I arrived on the 1st of October, 2014, to get a tour around this very familiar building in which I had previously spent five years as a student. I remember that Sandy, our freshly-postdoc tour guide, focused on showing us all the places where we could find coffee. For some reason, however, I never feel like drinking coffee when I'm at the university, so this knowledge has mostly gone to waste. On that day, I ended up in a wonderful office, with a shiny new laptop waiting for me. However, I did not have a key, a desk, or a desk chair, nor any idea how to start this PhD thingy.

Luckily, I had two great professors to help me along the way. Bart had many ideas on how to start, and Floris had lots of input and advice in the last years. Somehow, whether I had good or bad news, Floris never seemed unhappy to see me. Maybe he's just good at hiding it. But over time, we managed to shift the balance from lots of bad news, to lots of good news. I learned many things from him, about the workings of research, how to structure a paper, and most importantly, how to creatively combat our eternal enemy, the page limits. Floris knows all the tricks. I learned to be more expansive in my inveterately minimalistic explanations, more ostentatious in my presentations, and to be more judicious about my occasionally creative choice of words. Okay, I never learned that last one. Finally, I also learned not to copy his last-minute booking habits. I honestly don't think anyone could have been a better supervisor for me, and so compatible with my personality and way of working.

Throughout the past four years, I have had the pleasure of travelling quite a bit. From the exotic destinations, San Diego and Rio de Janeiro, to our

yearly trip to DBDBD, each one of them was great fun. I have had dinners next to an ocean to the tunes of a mariachi band, on a moving boat (which became slightly nauseating after some wine), and next to a dinosaur head. I am thankful for each of these unforgettable experiences.

Of course, while a PhD is at its core a very individual journey, the feeling of togetherness with all those colleagues on similar journeys is invaluable. Whether your research is in an up-phase or a down-phase, working in the Adrem group is always enjoyable. Over the years, we have had so many –for better or worse– unforgettable conversations over lunch. As I am sure you all immediately checked for this passage, here it comes. Many thanks to all of my past and present colleagues. I hope I did not forget anyone in this lengthy list: *Aida, Bart, Boris, Charlie, Cheng, Dahn, Dries, Elyne, Emmanuel, Emin, Hernán, Koen, Kris, Len, Lennert, Martin, Maarten, Maxime, Mozghan, Nicolas, Olivier, Pieter, Pieter, Reuben, Sandy, Sofie, Stefan, Stephen, Thomas, Tayena, Toon, Wim, Wout, and Youzhong.* No less than nine of these people have shared an office with me for a considerable time, even though I have not moved since day one. Either my company is enjoyable, or our airconditioning has something to do with that.

Of course, I was also not alone after I left the office at the end of the day. I would like to thank my parents for housing me, taking care of me, and unconditionally supporting me. My brother and sister, for looking after me ever since I was born. And, my three nieces, one tall and two small, for their, uhm ... Just being there was enough.

With all of your support, it never felt like my PhD adventure could go wrong. And, behold, it all went right.

Finally, I want to thank the members of my jury for investing their time in reading, critiquing, and generally improving this thesis. Chris Blondia as the chairman, Boris Cule, and the external members, Jef Wijsen and Paolo Papotti.

Thank you all!

Joeri Rammelaere
Antwerpen, 2018

Contents

Publications

- **Joeri Rammelaere**, and Floris Geerts. Revisiting Conditional Functional Dependency Discovery: Splitting the "C" from the "FD". In *Proc. Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2018)*, pages TBD, 2018.
- **Joeri Rammelaere**, and Floris Geerts. Explaining Repaired Data with CFDs. In *Proc. International Conference on Very Large Databases (PVLDB 2018)*, pages 1387–1399, VLDB Endowment, 2018.
- **Joeri Rammelaere**, Floris Geerts, and Bart Goethals. Cleaning Data with Forbidden Itemsets. In *Proc. International Conference on Data Engineering (ICDE 2017)*, pages 897–908, IEEE, 2017.
- **Joeri Rammelaere**, and Floris Geerts. Cleaning Data with Forbidden Itemsets [Journal Version]. Under revision for *Transactions on Knowledge and Data Engineering (TKDE 2018)*

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Over the past decades, the amounts of data being processed on computers have skyrocketed to astronomic proportions. Factors such as cheaper storage and increased connectivity have contributed to this data deluge. At the same time, this data has taken a central role in business, science, and society. The number of employees involved in the collection and processing of data has similarly increased, with data scientist being dubbed the sexiest job of this century [97]. Of course, only a small percentage of these employees are experts in computer science, and user-friendliness towards non-expert users is a key concern for data science tools.

While all this data is being generated, scraped and integrated at unprecedented rates, the quality control is often unable to keep up. Indeed, much of the data comes from unreliable sources, such as possibly faulty sensors and overworked humans. It is not possible to manually verify readings from thousands of sensors, or copy thousands of entries from one data source to another. Consequently, these massive amounts of data are becoming increasingly *dirty*, making it necessary to *clean* this data.

Clearly, if this dirtiness accumulates, then the data becomes unreliable for practical purposes. This is a huge problem for many businesses, with dirty data being estimated to cost the US economy hundreds of millions to trillions of dollars every year [61, 42, 62]. As such, there is a strong incentive to invest in solutions for cleaning the data. Apart from businesses suffering financial losses, the dirtiness of data also has a severe impact on areas such as data analytics, knowledge discovery from databases, and machine learning. These applications traditionally rely on large amounts of data, and if the data is typically dirty, then this may lead to wrong conclusions, faulty models, or spurious patterns. As the old adage goes, "garbage in equals garbage out". If the application is mostly harmless, for example playing Go, then this

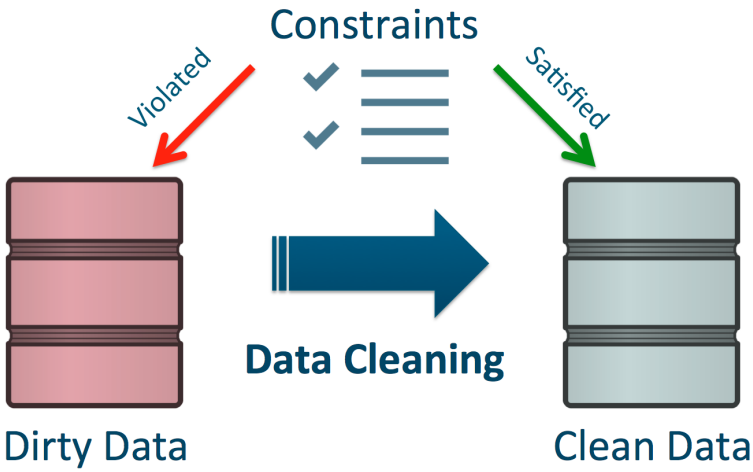


Figure 1.1: Schematic overview of constraint-based data cleaning.

might not be catastrophic. However, when the models are used for medical predictions, self-driving cars, or to assist decision-making for judges, then it is of paramount importance that the underlying data is reliable.

It is clear, then, that data quality has become a major problem in data management. The sheer volume of data makes it impossible for anyone to manually clean their data, and hence the demand for effective, semi or fully automatic methods to clean the dirty data is high. Dirtiness can come in many forms, including but not limited to incomplete data, duplicate data, obsolete data, and inconsistent data. In this dissertation, we focus on the latter problem of data inconsistency. In short, we are concerned with datasets in which combinations of values violate certain logical rules to which the data should adhere. This *constraint-based paradigm* is a focal point in data cleaning research, and is overviewed in the next section. Throughout this dissertation, we address the problem of discovering such constraints, or their violations, using methods akin to data mining. The field of data mining is briefly overviewed in Section 1.2. To conclude this chapter, the outline of the thesis is discussed.

1.1 Constraint-based Cleaning

In order to clean data in a principled way, a strong formal foundation for data cleaning is needed. In current data cleaning research, most methods are geared towards a constraint-based model. In such a model, constraints are provided in some logical formalism, and the data is either clean if all constraints are satisfied, or dirty if one or more constraints are violated. This is visualized in Figure 1.1. Examples of such constraints include, for example, demain restrictions or illegal value combinations. Given a set of constraints, the goal of data cleaning is then to modify the data such that all constraints are satisfied.

Many constraint formalisms have been introduced, and repair algorithms are in place to enforce these constraints. However, a frequently asked question is, "where do these constraints come from?". Indeed, in real-world situations, constraints are not readily available. The common answer to the question is that they are either supplied by experts, or automatically discovered from the data [44, 61]. In most real-world scenarios, however, experts are not available. Consequently, one has to rely on the automatic discovery of constraints from data. The discovery problem has received considerable research attention in recent years. Current discovery algorithms can infer which constraints currently hold on the data, and even apply statistical tests to validate these constraints. Still, it is not guaranteed that the discovered constraints make sense for the data. More importantly, clean data is typically not available, and hence constraints have to be discovered on precisely the data that one wants to clean. If the input is dirty, then this raises even more questions about the reliability of the discovered constraints. It boils down to the question: "which constraints are *semantically* valid?".

In this dissertation, we address the constraint discovery problem from three different angles. In Chapter 3, we generalize and improve existing constraint discovery algorithms. In Chapter 4, we involve the user in the constraint discovery process in order to ensure the discovered constraints are valid. In Chapters 5 and 6, we consider fully automatic repairing using a likeliness function to indicate which values are inconsistent.

1.2 Data Mining

Given large amounts of data, a user will naturally be interested in harvesting meaningful information from this data. This information can take the form of, for example, unexpected or insightful patterns, as well as predictive or descriptive models. The field of data mining, or more generally, knowledge discovery from data, is a research field concerned with extracting interesting information from (large) databases. Data mining methods combine techniques from machine learning, statistics, artificial intelligence and database systems. Data mining techniques have important applications in fields such as business analysis and fraud detection, as well as being used for the purpose of scientific discovery, such as in bio-informatics.

Within this dissertation, we make extensive use of a subfield of data mining called *pattern mining*. Pattern mining aims to discover interesting cooccurences or "items", or events, in a dataset. Typically, patterns are represented as *itemsets* or *association rules*. The quintessential application of pattern mining is supermarket customer data, such as in the example in Table 1.1. In this example, itemsets such as $\{Whisky, Bitters, Oranges\}$, and $\{Beer, Chips\}$, appear two times in this small dataset and are thus relatively frequent. Moreover, from these itemsets, rules can be derived such as "people who buy whisky and bitters, also buy oranges" (in order to make an Old Fashioned). The discovery of such rules and patterns can then be used by supermarkets to decide product pricing, promotions, store layout, and more.

Clearly, such techniques need to be tailored to the data on which they

Table 1.1: Example of a market basket database.

Transaction ID	Items
t_1	$\{Oranges, Beer, Chips\}$
t_2	$\{Whisky, Bitters, Oranges\}$
t_3	$\{Beer, Chips\}$
t_4	$\{Whisky, Bitters, Oranges\}$
t_5	$\{Whisky, Beer\}$

operate. In the example above, if the item "Whisky" was replaced with the names of specific brands, the association may not hold or be sufficiently frequent. This can be addressed by using, for example, ontological information. Moreover, instead of using supermarket basket data, we might also look for patterns in other types of data such as sequential data, episodic data or graph data. Specialized techniques exist for each of these types of data. In the context of this dissertation, we focus on data in a tabular format, also record data, where each row corresponds to a data object, and each column to an attribute.

In Chapters 4 and 5, we discover constraints using techniques based on association rule mining. In Chapters 6 and 7, we instead focus on mining itemsets, encoding unexpected co-occurences.

1.3 Organisation

In Chapter 2 we introduce basic concepts such as (conditional) functional dependencies and association rules, as well as notations used throughout the thesis, and discusses the various datasets used for experiments in the later chapters.

In Chapter 3 we discuss the most import related work within the general area of data cleaning. We focus mostly on overviewing the proposed constraint formalisms and cleaning systems.

In Chapter 4 we address the discovery problem for approximate conditional functional dependencies (CFDs). Within this chapter, we recast CFDs as a general form of association rules. Next, we present three distinct methodologies for discovering approximate CFDs, each combining functional dependency discovery and pattern mining in a different fashion. We show that the right choice of methodology, as well as search strategy, can have a substantial influence on the runtime of the discovery process. This chapter is based on Rammelaere and Geerts [93].

In Chapter 5 we present the XPLODE system, which discovers CFDs that *explain* modifications made manually by a user. Since the full set of discovered constraints is typically too large for inspection by a user, such *hybrid* discovery methods, where the user is somehow involved in the discovery process, have garnered research interest recently. Our XPLODE approach

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*. Guided by this explanation, data can then be cleaned using state-of-the-art CFD-based cleaning algorithms. We show that our method can typically infer the best explanation using a limited number of modifications, is efficient, and is robust to noise in the modifications. This chapter is based on Rammelaere and Geerts [94].

In Chapter 6 we introduce *forbidden itemsets*, which form the core of a fully automatic method for discovering and repairing errors. In this chapter, we introduce a *dynamic notion of data quality*, in which the data is clean if an error discovery algorithm, with set parameters, does not find any errors. Forbidden itemsets capture unlikely value co-occurrences in dirty data, which are often prime candidates for being errors. We provide an efficient algorithm for mining low lift forbidden itemsets, and derive properties of the lift measure to provide strong pruning. Our experiments show that the discovery of forbidden itemsets is efficient, and that such itemsets are adept at discovering errors. This chapter is based on a part of Rammelaere et al. [95].

In Chapter 7 we present the repair method for forbidden itemsets, discussing and proving how data can be repaired automatically such that no forbidden itemsets can be found. The efficiency of the repair method is contingent on the concept of *almost forbidden itemsets*, which are itemsets than can possibly become forbidden after a set number of modifications to the data. The repair algorithm modifies the dirty parts of the data by taking suggestions from the most similar clean parts of the data. Our algorithm is flexible in how such similarity is computed. Experiments show that mining almost forbidden itemsets is feasible, whereas repairs can be performed efficiently in parallel, and bring the dirty data closer to the ground truth. This chapter is based on the remaining parts of Rammelaere et al. [95].

In Chapter 8 we offer a retrospective on the work presented in this thesis, and a discussion on directions for future work.

1.4 Source Code

The source code of our algorithms, which are presented in chapters 4–7, has been made available on the CodeOcean platform for reproducible research:

- Joeri Rammelaere. Revisiting conditional functional dependency discovery [source code], June 2018. URL `https://codeocean.com/2018/06/20/discovering-conditional-functional-dependencies`
- Joeri Rammelaere. Xplode: Explaining repaired data with cfds [source code], June 2018. URL `https://codeocean.com/2018/06/10/xplode-colon-explaining-repaired-data-with-cfds`
- Joeri Rammelaere. Cleaning data with forbidden itemsets [source code], September 2018. URL `https://codeocean.com/2018/09/13/cleaning-data-with-forbidden-itemsets`

CHAPTER 2

Preliminaries

In the introductory chapter, we have already mentioned that we consider tabular datasets, and introduced the concepts of constraints, itemsets, and association rules. In this chapter, we formalize these concepts. At the end of the chapter, we discuss the various datasets used for experiments in the later chapters.

2.1 Concepts

Tabular Datasets

Formally, we consider relation schemas 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. We denote by $|D|$ the number of tuples in the instance D.

Given a set of attributes X in $\mathcal{A}$ and a tuple $t \in D$, we denote by $t[\mathsf{X}]$ the projection of that tuple on the attributes in X. Furthermore, 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.

Transaction Datasets

In the context of data mining, we consider instead *transaction* datasets $\mathcal{D}$ consisting of a finite collection of objects. An *object* o is a pair $\langle \mathsf{tid}, I \rangle$ where tid is an *object identifier*, e.g., a natural number, and I is an itemset. An *itemset* is a set of co-occuring *items*. An item takes the form (A, v), where A comes from the set $\mathcal{A}$ of *attributes* and v is a *value* from the domain $\mathsf{dom}(\mathsf{A})$ of A. An itemset contains at most one item for each attribute in $\mathcal{A}$. As such, a tuple t

corresponds simply to an object of maximal size $|\mathcal{A}|$, where each item (A, v) corresponds to the value $t[\mathsf{A}]$, and hence every tabular dataset can readily be converted into a transaction dataset. We make use of tabular datasets throughout Chapter 5, and utilize transactions datasets in Chapters 4, 6, and 7.

We define the value of an object $o = \langle \mathsf{tid}, I \rangle$ in attribute A, denoted by $o[\mathsf{A}]$, as the value v when $(\mathsf{A}, v) \in I$, and let $o[\mathsf{A}]$ be undefined otherwise. We denote by $|\mathcal{D}|$ the total number of objects in $\mathcal{D}$, and by $\mathcal{I}$ the set of all attribute/value pairs (A, v) in $\mathcal{D}$. Similar to tabular datasets, we denote by $\mathcal{D}[\mathsf{tid}]$ the object o with identier tid in $\mathcal{D}$, i.e., $o = \langle \mathsf{tid}, I \rangle$.

It is common in itemset algorithms to consider, at certain points during the algorithm, only a subset of the dataset $\mathcal{D}$, namely those objects that support a given itemset I. We call this set of objects the *projection* of $\mathcal{D}$ on I, denoted as $\mathcal{D}^I$:

$$\mathcal{D}^I = \{o = \langle \mathsf{tid}, J \rangle \in \mathcal{D} \mid I \subseteq J\}$$

Measures of Itemsets

An object $o = \langle \mathsf{tid}, I \rangle$ is said to *support* an itemset J if $J \subseteq I$, i.e., J is contained in I. The *cover* of an itemset J in $\mathcal{D}$, denoted by $\mathsf{cov}(J, \mathcal{D})$, is the set of tid's of objects in $\mathcal{D}$ that support J. As such, it is also called the *tidlist* of J.

$$\mathsf{cov}(J, \mathcal{D}) = \{\mathsf{tid} \mid \mathcal{D}[\mathsf{tid}] = \langle \mathsf{tid}, I \rangle s.t. J \subseteq I\}$$

The *support* of J in $\mathcal{D}$, denoted by $\mathsf{supp}(J, \mathcal{D})$, is defined as the number of tid's in its cover in $\mathcal{D}$:

$$\mathsf{supp}(J, \mathcal{D}) = |\mathsf{cov}(J, \mathcal{D})|$$

Similarly, the *frequency* of an itemset J in $\mathcal{D}$ is defined as the fraction of tid's in its cover:

$$\mathsf{freq}(J, \mathcal{D}) = \mathsf{supp}(J, \mathcal{D})|/|\mathcal{D}|$$

Moreover, we define $\mathsf{attrs}(I)$ as the set of attributes of all items in I:

$$\mathsf{attrs}(I) = \{\mathsf{A} \mid \exists v s.t. (\mathsf{A}, v) \in I\}$$

The union $I \cup J$ of two itemsets I and J is only defined over *compatible* itemsets, whose items cover a disjunct set of attributes. Formally, I and J are compatible if $\mathsf{attrs}(I) \cap \mathsf{attrs}(J) = \emptyset$.

Association Rules

As *association rule* between two itemsets I and J, denoted as $I \to J$, indicates that the occurrence of itemset X implies the occurrence of itemset Y within the same object. This implication is typically quantified using an *interestingness measure*. Through this dissertation, we make use of two such measures, *confidence* and *lift*.

The confidence of an association rule between itemsets I and J, is the fraction of objects in I's cover that support J:

$$\mathsf{conf}(I \to J, \mathcal{D}) = \mathsf{supp}(I \cup J, \mathcal{D}) / \mathsf{supp}(I, \mathcal{D})$$

The lift between two itemsets I and J compares the actual number of co-occurences of I and J with the expected number of co-occurences if I and J are independent of each other. A lift higher than 1 indicates a positive correlation, while a left less than 1 indicates a negative correlation. More formally, the lift of the association rule $I \to J$ is defined as:

$$\mathsf{lift}(I \to J, \mathcal{D}) := \frac{\mathsf{freq}(I \cup J, \mathcal{D})}{\mathsf{freq}(I, \mathcal{D}) \times \mathsf{freq}(J, \mathcal{D})}$$

We note that the lift measure is symmetrical, unlike confidence, and hence $\mathsf{lift}(I \to J, \mathcal{D}) = \mathsf{lift}(J \to I, \mathcal{D})$.

Vertical Data Layout

We sometimes represent a transaction dataset $\mathcal{D}$ in a *vertical data layout*, denoted by $\mathcal{D}\downarrow$. More formally, we define:

$$\mathcal{D}\downarrow = \left\{ (i, \mathsf{cov}(\{i\}, \mathcal{D})) \mid i \in \mathcal{I}, \mathsf{cov}(\{i\}, \mathcal{D}) \neq \emptyset \right\}$$

Clearly, one can freely go from $\mathcal{D}$ to $\mathcal{D}\downarrow$, and vice versa.

Conditional Functional Dependencies

Conditional functional dependencies (CFDs) [45] are constraints which are frequently used in the field of data cleaning. A CFD φ over a schema R is a pair $(\mathsf{X} \to \mathsf{A}, t_p)$, where:

(i) X is a set of attributes in $\mathcal{A}$, and A is a single attribute in $\mathcal{A}$;

(ii) $\mathsf{X} \to \mathsf{A}$ is a standard functional dependency (FD), called the embedded FD of φ; 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 $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ is called variable if $t_p[\mathsf{A}] =$ '_', otherwise it is called *constant*. For constant CFDs, the pattern tuple $t_p[X]$ is assumed to consist of constants only. On the other hand, a regular FD is then a (variable) CFD with t_p consisting solely of variables '_'. As such, constant CFDs enforce consistency within individual tuples, whereas variable CFDs enforce consistency among two or more tuples.

Semantics of CFDs

The semantics of a CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ on an instance D are 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 $t[\mathsf{B}] = t_p[\mathsf{B}]$. In other words, the tuple t needs to have the same value as the pattern t_p for all attributes in B, except if t_p contains a wildcard for the attribute.

The tuple t *violates* a *variable* CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ if it matches the pattern, i.e., $t[\mathsf{X}] \asymp t_p[\mathsf{X}]$, and there exists another tuple t' in D such that $t[\mathsf{X}] = t'[\mathsf{X}]$,

while $t[\mathsf{A}] \neq t'[\mathsf{A}]$. In other words, there exist multiple tuples matching φ, with identical values on the attributes X, yet distinct values for the attribute 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 $t[\mathsf{A}] \neq t_p[\mathsf{A}]$. In other words, the tuple has identical values to the pattern tuple on the attributes X, but different values on the attribute A.

The set of all tids of tuples in D that violate a CFD φ is denoted by $\mathsf{VIO}(\varphi, D)$. If $\mathsf{VIO}(\varphi, D) = \emptyset$, then D *satisfies* φ, denoted by $D \models \varphi$.

Measures of CFDs

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 in D that match the pattern t_p of φ on the set of attributes X:

$$\mathsf{supp}(\varphi, D) = |\{t \in D : t[\mathsf{X}] \asymp t_p[\mathsf{X}]\}|$$

The CFD φ is then called *frequent* in D, if $\mathsf{supp}(\varphi, D) \geqslant \delta$, where δ is a user-specified support threshold.

In line with the commonly used notion of confidence for approximate FDs [60], we define the *confidence* of a CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ in D as:

$$\mathsf{conf}_{\mathsf{FD}}(\varphi, D) = 1 - \frac{|D'|}{\mathsf{supp}(\varphi, D)}$$

Here, $D' \subseteq D$ denotes a minimal subset, in terms of cardinality, such that $D \setminus D' \models \varphi$. This definition is well-suited to variable CFDs, where the set $\mathsf{VIO}(\varphi, D)$ contains all tuples that *together* violate the CFD, since variable CFDs apply to two or more tuples. However, the individual tuples in the violation set 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, $|D'| = |\mathsf{VIO}(\varphi, D)|$, and hence this definition of confidence reduces to the standard confidence of an association rule. This confidence measure is also called g_3 in Kivinen and Mannila [66]. We observe that $\mathsf{conf}_{\mathsf{FD}}(\varphi, D) = 1$ means that $D \models \varphi$ and $\mathsf{conf}_{\mathsf{FD}}(\varphi, D) = 0$ means that all tuples matching $t_p[\mathsf{X}]$ need to be altered. Note that this can only occur for constant CFDs, since a variable CFD is trivially satisfied if it applies to a single tuple.

The CFD φ is called *exact* if $\mathsf{conf}_{\mathsf{FD}}(\varphi, D) = 1$, and *approximate* otherwise. Given a user-supplied threshold ε, we say a CFD is *confident* in D if $\mathsf{conf}_{\mathsf{FD}}(\varphi, D) \geqslant 1 - \varepsilon$. Throughout this dissertation, we consider CFDs that are both frequent and confident, which we call (ε, δ)-CFDs.

Definition 1 ((ε, δ)-CFD)**.** A CFD φ on a tabular database instance D is an (ε, δ)-CFD if $\mathsf{conf}_{\mathsf{FD}}(\varphi, D) \geqslant 1 - \varepsilon$ and $\mathsf{supp}(\varphi, D) \geqslant \delta$. □

CFDs as Association Rules

In Chapter 4, we will present CFD discovery algorithms using concepts from association rule mining. We now formalize the relationship between CFDs

and association rules. A CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ over a tabular dataset D can be written compactly in the form of an association rule over the corresponding transaction dataset $\mathcal{D}$. To incorporate the semantics of a CFD into this format, we consider an extension of itemsets, where an item i can also take the form $(\mathsf{A}, \text{`\_'})$. In other words, we model CFD semantics by including wildcard-items as a special type of item. We can now write φ as $I \to j$, between an itemset I and a single item j, where the itemset I is of the form $I = \bigcup_{\mathsf{B} \in \mathsf{X}} \{(\mathsf{B}, t_p[\mathsf{B}])\}$, and the single item $j = (\mathsf{A}, t_p[\mathsf{A}])$.

The CFD discovery algorithms considered in chapters 4 and 5 are based on the concept of *equivalence partitions*, as introduced in Cosmadakis et al. [36], and used in the Tane [60] and CTane [46] algorithms. More specifically, given an itemset I consisting of attribute-value pairs, we say that two tuples s and t in D are *equivalent relative to* I if, for all $(\mathsf{B}, v) \in I$, $s[\mathsf{B}] = t[\mathsf{B}] \asymp v$. For a tuple $s \in D$, $[s]_I$ denotes the *equivalence class* consisting of the tids of all tuples $t \in D$ that are equivalent with s relative to I. The *(equivalence) partition of* I, denoted by $\Pi(I)$, is the collection of $[s]_I$ for $s \in D$ [1]. For a single constant item, $\Pi((\mathsf{A}, v)) = \{\mathsf{cov}((\mathsf{A}, v), D)\}$, i.e., it consists of (A, v)'s tidlist. For a single variable item, $\Pi((\mathsf{A}, \_)) = \{\mathsf{cov}((\mathsf{A}, v), D) \mid v \in \mathsf{dom}(\mathsf{A})\}$, i.e., it consists of all tidlists grouped together with regards to the A-values of the corresponding tuples. For an itemset I, $\Pi(I) = \bigcap_{i \in I} \Pi(i)$ in which equivalence classes are pairwise intersected. The *size* of $\Pi(I)$, denoted by $|\Pi(I)|$, is the number of equivalence classes in $\Pi(I)$. We use $\|\Pi(I)\|$ to denote the number of tids in $\Pi(I)$, equal to the support of I. Finally, we note that the CFD $I \to j$ holds if and only if $|\Pi(I)| = |\Pi(I \cup \{j\})|$.

2.2 Example

We illustrate the concepts defined above using the "play tennis" example dataset from [89], shown in Table 2.1. This dataset contains, for instance, the items $(\mathsf{Windy}, \mathrm{false})$, and $(\mathsf{Play}, \mathrm{play})$. The itemset $\{(\mathsf{Windy}, \mathrm{false}), (\mathsf{Play}, \mathrm{play})\}$ is supported by objects $t_3, t_4, t_5, t_9, t_{10}$, and t_{13}. Hence, it has a support of 6, and a frequency of $6/14 \approx 0.43$. The superset $\{(\mathsf{Windy}, \mathrm{false}), (\mathsf{Play}, \mathrm{play}), (\mathsf{Humidity}, \mathrm{normal})\}$ has a support of 4. As such, we can obtain the association rule

$$\{(\mathsf{Windy}, \mathrm{false}), (\mathsf{Play}, \mathrm{play})\} \to (\mathsf{Humidity}, \mathrm{normal})$$

This association rule has a confidence of $4/6 \approx 0.67$, and a lift of

$$\frac{4/14}{6/14 \times 7/14} \approx 1.33,$$

indicating a positive correlation between the occurrence of the itemsets $\{(\mathsf{Windy}, \mathrm{false}), (\mathsf{Play}, \mathrm{play})\}$ and $\{(\mathsf{Humidity}, \mathrm{normal})\}$.

Let us now turn our attention to CFDs. A possible approximate CFD φ on this dataset is $\{(\mathsf{Windy}, \mathrm{false}), (\mathsf{Outlook}, \_)\} \to (\mathsf{Play}, \_)$. For brevity of no-

[1] Strictly speaking this is only a partition of D when I contains variable items $(\mathsf{A}, \_)$.

tation, let $I = \{(\text{Windy}, \text{false}), (\text{Outlook}, \_), (\text{Play}, \_)\}$ and $j = (\text{Play}, \_)$. The relevant equivalence partitions are then

$$\Pi(I \setminus \{j\}) = \{\{1,8,9\},\{3,13\},\{4,5,10\}\}, \text{ and}$$
$$\Pi(I) = \{\{1,8\},\{9\},\{3,13\},\{4,5,10\}\}.$$

The sizes of these equivalence partitions are $|\Pi(I \setminus \{j\})| = 3$ and $|\Pi(I)| = 4$, and both partitions have support $||\Pi(I \setminus \{j\})|| = ||\Pi(I)|| = 8$. The supported tuples t, i.e., where $t[\text{Windy}] = \text{false}$, are shown in Table 2.1, with different colors corresponding to the different equivalence classes in $\Pi(I)$.

The CFD can be made to hold exactly by removing the tuple with tid 8, such that $\Pi(I \setminus \{j\}) = \Pi(I)$. Hence, the minimal subset D' contains only the tuple with tid 8, and $|D'| = 1$. We now have that

$$\text{conf}_{\text{FD}}(I \to j, D) = 1 - (|D'|/||\Pi(I)||) = 1 - (1/8) = 0.875$$

Finally, the set of violations $\text{VIO}(\varphi, D)$ contains the tuples $\{t_1, t_8, t_9\}$.

Table 2.1: Play tennis dataset [89].

tid	Outlook	Temperature	Humidity	Windy	Play
t_1	sunny	hot	high	false	dont
t_2	sunny	hot	high	true	dont
t_3	overcast	hot	high	false	play
t_4	rain	mild	high	false	play
t_5	rain	cool	normal	false	play
t_6	rain	cool	normal	true	dont
t_7	overcast	cool	normal	true	play
t_8	sunny	mild	high	false	dont
t_9	sunny	cool	normal	false	play
t_{10}	rain	mild	normal	false	play
t_{11}	sunny	mild	normal	true	play
t_{12}	overcast	mild	high	true	play
t_{13}	overcast	hot	normal	false	play
t_{14}	rain	mild	high	true	dont

2.3 Datasets

To conclude this chapter, we present an overview of the datasets used throughout the dissertation. Most of these were taken from the UCI Machine Learning Repository [39]. The statistics of the datasets are shown in Table 2.2.

Abalone is a UCI dataset containing measurements of abalones, a kind of marine snail, with the goal of predicting their age.

Table 2.2: Statistics of datasets used throughout the dissertation.

Dataset	Nr. Tuples	Nr. Attributes	Nr. Distinct Items
Abalone	8354	9	6077
Adult	48842	11	202
Censusincome	199523	12	234
CreditCard	30000	12	216
Ipums	70187	32	364
Mushroom	8124	23	119
Nursery	12960	9	32
Soccer	200000	10	767
SP500	245148	7	136718

Adult is a UCI dataset containing census data, with the goal of predicting a person's income. Some discretization was performed on the Age attribute.

Censusincome is a UCI dataset containing weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau, with the goal of predicting household income. We have retained 12 categorical attributes.

CreditCard is a UCI dataset containing information about credit card users in Taiwan, with the goal of predicting payment defaults. We have retained 12 categorical attributes [113], since the numerical attributes are mostly unique and hence not imperative for the techniques used throughout this dissertation.

Ipums is a UCI dataset containing unweighted PUMS census data from the Los Angeles and Long Beach areas [99].

Mushroom is a UCI dataset describing various mushrooms, with the goal of predicting whether they are edible.

Nursery is a UCI dataset containing applications for nursery schools in Slovenia, with the goal of predicting the evaluation of the application.

Soccer is a synthetic dataset from Arocena et al. [8] containing details about soccer players in various seasons [2].

SP500 is a dataset from Chu et al. [32] containing information about stock trading, from Standard & Poor's.

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

CHAPTER 3

Related Work

In this chapter, we discuss the most relevant work related to our general problem setting of constraint discovery for declarative data quality. For recent overviews of the research field of data cleaning, we refer to Fan and Geerts [44] and Ilyas and Chu [61]. Current challenges and are discussed in Abedjan et al. [2], Chu et al. [35], and Meduri and Papotti [81], among others. In the subsequent chapters, we discuss additional related work, specifically related to the contents of that chapter.

3.1 Constraint Discovery

Within the field of constraint-based data quality, a variety of formalisms have been proposed. We briefly overview those types of constraints and cleaning rules that are most related to our work. Some of these are further discussed in Liu et al. [72].

- **Functional Dependencies** (FDs) [3] are a staple in database theory, and are traditionally used for database design and normalization. Clearly, an FD can also detect inconsistencies between tuples, and as such, FDs are also used for data cleaning [44]. However, the applicability of FDs to data cleaning is limited since they need to hold on the entire dataset.

- **Conditional Functional Dependencies** (CFDs) were introduced in Fan et al. [45] as an extension of FDs, relaxing the aforementioned limitation of FDs. CFDs add a conditional pattern to a standard FD, stating that the FD only holds on the subset of the data that matches this conditional pattern. Moreover, using these patterns, CFDs can also be used to detect inconsistencies between values in a single tuple. Extensions of CFDs are discussed in Bravo et al. [22].

- **Denial Constraints** (DCs) are a universally quantified first order logic formalism, and more expressive than FDs and CFDs, supporting operators such as inequalities as well as normal equalities. In the context of data cleaning [31], a DC specifies combinations of logical predicates that are illegal, either within a single tuple or across multiple tuples. However, the expressiveness of DCs comes at the price of a slow discovery process [32]. While the recently proposed Hydra algorithm by Bleifuß et al. [17] is orders of magnitude faster than previous work for discovering exact DCs, it has not yet been applied to *approximate* DCs, which are typically required in a cleaning context. It is still unclear how an extension to approximate DCs would impact the reported runtimes.

- **Matching Dependencies** (MDs), introduced in Fan [43], are constraints that involve a similarity function. In other words, the constraint does not only apply to data entries which *exactly* match a given pattern, put also those that are within a certain similarity. Likewise, a violation of the rule ensues when a certain value in the data is not similar enough to the specified target value for the constraint. As with FDs, MDs have been extended with conditional patterns in Wang et al. [106]. Similar to MDs are **Metric Dependencies**, proposed in Koudas et al. [68], which also consider a similarity metric, but only on the consequent of a rule, i.e., data must still exactly match the pattern of the rule in order for the rule to apply.

- **Differential Dependencies** (DDs) were introduced in Song and Chen [100], as a generalization of matching and metric dependencies. Instead of requiring a certain similarity between values, DDs use distances (defined using differential functions). As such, DDs can also express orderings on values, or specify a required distance between values (as opposed to a required similarity), for instance. DDs have also been extended with constant patterns, in Kwashie et al. [70].

- **Fixing Rules** were proposed in Wang and Tang [104] to not only detect inconsistencies between different values in data, but also provide a principled way of detecting which of the inconsistent values should be update, and how this update should be done. Since fixing rules encode expert knowledge, they are provided up front by a user; it is unclear how such rules could be discovered automatically. In a similar vein, Interlandi and Tang [63] introduces **Sherlock Rules**.

- **Editing Rules**, used in Fan et al. [47], take a different approach by making use of *master data* for repairing. By leveraging this master data, which is guaranteed to be correct, editing rules aim to generate repairs that are also guaranteed to be correct. However, master data is not always available, and in this dissertation we consider rule discover without the use of master data.

- In certain contexts, data values might be syntactically different yet semantically equivalent, possible at a higher level of abstraction. **Ontology Functional Dependencies** (OFDs), introduced in Baskaran et al.

[11], handle this situation by adding ontological information into the rule discovery process. More specifically, distances between values are computed at the ontological level, instead of purely on the values themselves. OFDs are an enhancement of **Roll-up Dependencies**, proposed in Wijsen et al. [109].

- Instead of considering tabular datasets, recent work by Arioua and Bonifati [7] and Ortona et al. [83] concerns data quality rules over **knowledge bases** (KBs). An important challenge when considering rules over KBs, is the open world assumption: in contrast to traditional databases, KBs are typically not assumed to be complete, and hence facts that are missing from the data are not necessarily wrong. Arioua and Bonifati [7] presents a method for repairing KBs under **contradiction detecting dependencies**, a subset of DCs similar to the Forbidden Itemsets presented in Chapter 6. The goal is to remove all contraditions from the KB by asking a minimal number of questions to a user. In Ortona et al. [83], the KB is used to generate sets of positive and negative examples (where negative examples correspond to errors). Rules are then discovered that cover many positive examples, and few negative examples.

3.2 Data Repairing

In the data repairing step, it is typically assumed that the constraints are known and fixed, and hence, the errors in the data are known. Various systems exist that modify data in such a way to remove the errors. Many of the repair algorithms are based on a chase algorithm [13, 74] or a cost minimization approach as in Bohannon et al. [18]. We overview some of the most important cleaning systems:

- **Llunatic**, presented in Geerts et al. [51], is a cleaning framework. Instead of hardcoding a repair strategy, Llunatic makes use of customizable strategies for selecting *preferred values*, which in turn specify how inconsistencies are to be repaired. Consequently, Llunatic offers a uniform framework for repairing in the presence of different kinds of constraints.

- **Nadeef**, discussed in Ebaid et al. [41] and Dallachiesa et al. [38], is an *extensible* framework for cleaning, intended to semi-automate the entire process. It contains a core functionality for detecting errors, combining various rule types, and subsequently cleaning these errors. At each part of the cleaning pipeline, users can insert custom code to tailor the process.

- **Holistic Data Cleaning** is presented in Chu et al. [31] as a repair method based on Denial Constraints. Based on these given DCs, the Holistic method computes a conflict hypergraph, capable of handling different types of constraints, possibly with numeric operators. A holistic repair algorithm is given, which is based on heuristics and

optimizes an arbitrary objective function. The conflict hypergraph is used to reduce the number of cells that needs to be looked at to guarantee a successful repair.

- **HoloClean**, introduced in Rekatsinas et al. [96], is a *probabilistic* cleaning framework. It takes again a set of constraints as given, and builds a probabilistic model to repair the violations of these constraints. These repairs are thus based on the statistical properties of the data. HoloClean can also optionally incorporate external data, such as master data.

- **Dance**, introduced in Assadi et al. [9], makes use of *domain experts* during the cleaning process. Dance takes a set of constraints as input, and builds a graph of the violations in order to discover the "most suspicious" tuples. These tuples are then presented to the expert, in order to clean the data with limited user interaction.

- **GDR**, or Guided Data Repair, was introduced in Yakout et al. [110]. Similar to Dance, GDR also assumes a given set of constraints, and employs user interaction in order to perform correct repairs. To minimize the number of required user interactions, GDR uses the user's feedback as input to an active learning algorithm, which then ranks the possible updates before presenting them to the user again.

- **BigDansing**, presented in Khayyat et al. [65], is focused on the scalability of the actual repair process. Constraints are given, and both the violation detection phase and the repairing phase are performed on a distributed platform to improve performance.

- **SCARE**, introduced in Yakout et al. [111], uses statistical machine learning techniques to automatically clean a dirty dataset. By using available master data or constraints, SCARE selects a subset of the data that is, with a very high likelihood, clean. A statistical model is then trained on this clean subset, learning the distribution of "clean data". Subsequently, the remaining part of the data is minimally adjusted so as to maximally align with the learned model of the data distribution.

- **Katara**, presented in Chu et al. [34], takes a different approach to traditional rule-based cleaning systems. Instead of relying on heuristics or input from a single user, Katara employs knowledge bases and crowdsourcing. By using these as known facts, Katara then discovers the errors in a given dirty dataset, and automatically repairs them.

CHAPTER 4

Revisiting Conditional Functional Dependency Discovery

4.1 Introduction

In the introduction, we have outlined the constraint-based approach to data quality. Among the variety of proposed formalisms, a popular type of constraint are conditional functional dependencies (CFDs).

In this chapter, we consider CFDs recast as an extension of association rules, as formalized in Section 2.1. We discuss CFD discovery from a more general perspective, and distinguish *three general methodologies* for discovering *confident* CFDs[1], as typically used for data cleaning, based on distinct ways of combining FD discovery with itemset mining. The first methodology is used by the CTane algorithm [46], and performs an integrated traversal of the lattice containing all possible CFDs. Additionally, we introduce two new methodologies, which explicitly consider CFD discovery as *a combination of FD discovery and pattern mining*. We introduce an *itemset-centric* approach, where patterns are mined at the top level, and FDs are subsequently discovered on the corresponding subsets of the data; and an *FD-centric* approach, which at the top level traverses the search space of FDs, and then mines those patterns for which the FD holds, generalizing the approach taken in FindCFD [29]. Moreover, in the FD-centric approach, we identify techniques for speeding up the pattern mining process, using information from the FD discovery process at the top level. A high-level overview of the three methodologies is shown in Figure 4.1.

[1] Other interestingness measures can be plugged in, if they can be computed from equivalence partitions. This is the case for most popular measures.

CFD Discovery Methodologies		
Integrated	**Itemset-First**	**FD-First**
- Discover all CFDs	- Discover all Frequent Itemsets - Foreach itemset: -- Discover FDs	- Discover all FDs - Foreach FD: -- Discover frequent patterns

Figure 4.1: High-level overview of the three methodologies for the discovery of CFDs.

Both new methodologies are described in a flexible way, enabling the use of *any* FD discovery method based on *equivalence partitions*, and *any* itemset mining method based on *tidlists*, for each of the separate steps. As such, the methodologies we describe, represent in fact a *family* of algorithms. This has as a direct advantage that *CFD discovery can benefit directly from advances in FD and itemset discovery*.

4.1.1 Summary of Contributions

1. We introduce three general methodologies for discovering frequent, approximate CFDs. Each of these methodologies can be thought of as a family of concrete algorithms. (Section 4.3)

2. We derive the worst-case time complexity of our CFD discovery algorithms. (Section 4.3.5)

3. We present a general pruning strategy for CFDs, such that each methodology can use an arbitrary strategy for traversing the search space of CFDs, e.g., breadth-first or depth-first. Note that both CTane and FindCFD were originally presented using a breadth-first strategy, because of pruning. (Section 4.3.6)

4. We show experimentally that both of our proposed methods typically outperform the integrated approach to CFD discovery, which is used by CTane. The FD-centric approach performs substantially better in most cases, especially on data with a higher number of attributes. We also identify situations in which the itemset-centric approach provides the best performance, namely when using a very low minimum support threshold. Moreover, the appropriate use of depth-first search strategies further improve runtime for the different methodologies. (Section 4.5)

4.2 Problem Statement

In this chapter, we address the problem of discovering approximate CFDs with high support:

PROBLEM: Approximate, Frequent CFD Discovery	
INPUT:	Instance D of a schema R, confidence threshold ε and support threshold δ.
OUTPUT:	All CFDs φ over R with: $\mathsf{supp}(\varphi, D) \geqslant \delta$ and $\mathsf{conf}_{\mathsf{FD}}(\varphi, D) \geqslant 1 - \varepsilon$.

Since this chapter is based on techniques from association rule mining, we convert the tabular database instance D into a transaction dataset $\mathcal{D}$. In the remainder of this chapter, we make use of $\mathcal{D}$.

4.3 Three approaches for CFD Discovery

We present three general approaches for the discovery of approximate CFDs with high supports. All of the approach are based on a *lattice traversal*, as commonly used in itemset and association rule discovery algorithms. The approaches differ in the way that this (itemset) search lattice is explored. First, we generalize the *integrated* approach, used in the existing CTane algorithm [46], in which a combined search lattice of constant and variable ('_') patterns is traversed at once. For the other two, new approaches, we *decouple* the lattices for constant and variable patterns. We present the *Itemset-First* approach, followed by the *FD-First* approach. Both of these approaches consist of two separate algorithms, which either explore a lattice containing only constant patterns, or containing only variable patterns. After discussing the three methodologies, we derive the general time complexity of CFD discovery.

As mentioned in the introduction, to add further flexibility to each of the three methods, we describe our algorithms *independent* from the search strategy (breadth or depth-first) used. To achieve uniform pruning across all approaches and search strategies, we present pruning strategies based on a generalization of free itemsets [21] and a lookup table.

4.3.1 Integrated CFD Discovery

We start by describing the integrated approach MINE-INTEGRATED for discovering CFDs, as implemented by CTane [46]. Its pseudocode is shown in Algorithm 1.

For each of our approaches, we represent the search space of all itemsets can be a *power set lattice*, with the empty itemset at the bottom and the set containing all items at the top. We call the elements of the lattice *nodes*, and

Algorithm 1 Integrated CFD discovery algorithm

```
1:  procedure MINE-INTEGRATED(𝒟, δ, ε)
2:      ℒ^INT ← {(A, v) | A ∈ 𝒜, v ∈ dom(A) ∪ {_}, supp((A, v), 𝒟) ⩾ δ}
3:      Compute Π({i}, D) for all i ∈ ℒ^INT
4:      Initialize fringe with ℒ^INT depending on search strategy
5:      Σ ← ∅
6:      while fringe not empty do
7:          I ← POP(fringe)
8:          for all j ∈ I do
9:              if conf_FD(I \ {j} → j, 𝒟) ⩾ 1 − ε then
10:                 Σ ← Σ ∪ {I \ {j} → j}
11:         insert children of I into fringe if supp(I, 𝒟) ⩾ δ
12:     return Σ
```

for a given itemset I, we denote all nodes in the lattice corresponding to itemsets $J \supset I$ as the *children* of I. Conversely, all nodes corresponding to itemsets $K \subset I$ are called the *parents* of I. A *level* of the lattice consists of the set of all nodes of a given size, e.g., the first level contains all itemsets of length 1.

Algorithms based on the integrated methodology traverse the entire search lattice for CFDs, consisting of both constant *and* variable patterns:

Definition 2 (Integrated Search Lattice)**.** Consider a database instance $\mathcal{D}$. The search lattice $\mathcal{L}^{\text{INT}}$ for the integrated methodology consists of all itemsets I in the powerset $\mathcal{P}(\mathcal{I} \cup \{(\mathsf{A}, \text{'\_'}) \mid \mathsf{A} \in \mathcal{A}\})$, such that I contains at most one item per attribute, i.e., $|\mathsf{attrs}(I)| = |I|$. □

The first level $\mathcal{L}^{\text{INT}}$ of this lattice is initialized on line 2 of the pseudocode. For each singleton item, its equivalence partition is computed from the data; only sufficiently frequent constant items are retained.

The lattice is subsequently traversed, on lines 6–11, typically in either a breadth-first or depth-first manner [2]. Regardless of the choice of traversal, we refer to the set of current lattice elements considered as the *fringe*.

Whenever an itemset I in the fringe is visited (line 7), all CFDs of the form $I \setminus \{j\} \to j$, for $j \in I$, are generated, and their confidence is computed from the equivalence partitions $\Pi(I \setminus \{j\})$ and $\Pi(I)$. If the confidence exceeds the threshold, then the CFD is added to the result Σ. An efficient algorithm for computing confidence is presented in Tane [60], and is based on the *error* of an equivalence class.

Definition 3 (Refinement of an Equivalence Partition)**.** Given an itemset I and a single item j, for all $\mathsf{eq} \in \Pi(I \setminus \{j\})$, we define the *refinement* of eq in $\Pi(I)$, denoted by $\Pi(I)^{\mathsf{eq}}$, as $\{\mathsf{eq}' \in \Pi(I) \mid \mathsf{eq}' \subset \mathsf{eq}\}$. □

In other words, $\Pi(I)^{\mathsf{eq}}$ contains all equivalence classes over I that match the same (constant) pattern as eq on the attributes $I \setminus \{j\}$. This concept is used to define the error:

[2] The CTane algorithm as presented in [46] employs a breadth-first traversal.

Definition 4 (Error of an Equivalence Partition)**.** Given an itemset I and a single item j, we define

$$\mathsf{error}(\mathsf{eq}, \Pi(I)) = ||\Pi(I)^{\mathsf{eq}}|| - \max_{\mathsf{eq}' \in \Pi(I)^{\mathsf{eq}}} |\mathsf{eq}'|. \qquad \square$$

Generalizing the argument given in [60] for variable patterns to arbitrary (constant and variable) patterns, the confidence can then be computed as:

Definition 5 (Computational formula for the confidence of a CFD)**.**

$$\mathsf{conf}_{\mathsf{FD}}(I \setminus \{j\} \to j) = 1 - \frac{\sum_{\mathsf{eq} \in \Pi(I \setminus \{j\})} \mathsf{error}(\mathsf{eq}, \Pi(I))}{\mathsf{supp}(I \setminus \{j\})}. \qquad \square$$

Example 1. We consider the CFD $\big\{(\mathsf{Windy}, \mathsf{false}), (\mathsf{Outlook}, \_)\big\} \to (\mathsf{Play}, \_)$ from the Tennis dataset in Table 2.1. To simplify notation, we let $I = \big\{(\mathsf{Windy}, \mathsf{false}), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\big\}$ and $j = (\mathsf{Play}, \_)$. We compute the error for each of the 3 equivalence classes in $\Pi(I \setminus \{j\})$: $\{1,8,9\}, \{3,13\}$, and $\{4,5,10\}$. For $\mathsf{eq} = \{3,13\}$ and $\mathsf{eq} = \{4,5,10\}$, we have $|\Pi(I)^{\mathsf{eq}}| = 1$, since the objects within these equivalence classes have the same values for attribute Play. Hence, there is only one $\mathsf{eq}' \in \Pi(I)^{\mathsf{eq}}$, and $||\Pi(I)^{\mathsf{eq}}|| = \max_{\mathsf{eq}' \in \Pi(I)^{\mathsf{eq}}} |\mathsf{eq}'|$, leading to an error of 0. This leaves us with $\mathsf{eq} = \{1,8,9\}$, for which $\Pi(I)^{\mathsf{eq}} = \big\{\{1,8\}, \{9\}\big\}$. Indeed, this is the equivalence class containing the violations of the CFD. We compute the error as

$$\mathsf{error} = ||\big\{\{1,8\}, \{9\}\big\}|| - \max(|\{1,8\}|, |\{9\}|) = 1,$$

resulting in a confidence of $1 - (\mathsf{error}/||\Pi(I)||) = 1 - (1/8) = 0.875$. $\diamondsuit$

Finally, if I is sufficiently frequent, the children of I in the lattice are generated and inserted into the fringe (line 11). This is done by joining I with all itemsets J in the fringe that are (i) at the same level in the lattice, i.e., $|J| = |I|$; and (ii) such that J and I differ in only one item. A child M is then obtained as $I \cup J$, and $\Pi(M)$ is computed by intersecting $\Pi(I)$ with $\Pi(J)$. The Tane algorithm provides a linear algorithm for computing such an intersection, making use of a lookup table. Using a similar technique, confidence can be computed in linear time. The algorithm for intersection is briefly recapped in Section 4.3.4.

4.3.2 Itemset-First Discovery

The second, and new, approach to CFD discovery starts with an itemset mining step. The pseudocode of algorithm MINE-ITEMSET-FIRST is shown in Algorithm 2.

For the Itemset-First approach, we consider a search lattice containing only items with constant values:

Definition 6 (Itemset-First Search Lattice)**.** Consider a database instance $\mathcal{D}$. The search lattice $\mathcal{L}^{\text{IT-F}}$ for the Itemset-First methodology consists of all itemsets I in the powerset $\mathcal{P}(\mathcal{I})$, such that I contains at most one item per attribute, i.e., $|\mathsf{attrs}(I)| = |I|$. $\square$

The first level of the search lattice $\mathcal{L}^{\text{IT-F}}$ is initialized on line 2. Since we consider only items with constant values, we only require the cover of each item in $\mathcal{L}$ (recall that the equivalence partition of a constant item corresponds to its cover). The lattice is again traversed using an arbitrary search strategy and generated itemsets are inserted into the fringe.

When visiting an itemset I in this approach, we initialize a separate FD searching algorithm (line 8). The item lattice for this FD search ($\mathcal{L}_{\text{FD}}^{\text{IT-F}}$) now consists only of those items in D with a variable pattern ('_'), and whose attribute is not already present in $\mathsf{attrs}(I)$:

Definition 7 (Itemset-First Sublattice for FD-search)**.** Consider a database instance $\mathcal{D}$ and an itemset I being processed in the Itemset-First methodology. The sublattice $\mathcal{L}_{\text{FD}}^{\text{IT-F}}$ for the FD-search on itemset I consists of all itemsets J in the powerset $\mathcal{P}(\{(\mathsf{A}, \text{'\_'}) \mid \mathsf{A} \in (\mathcal{A} \setminus \mathsf{attrs}(I))\})$, such that J contains at most one item per attribute, i.e., $|\mathsf{attrs}(J)| = |J|$. □

In other words, we wish to extend the constant pattern I with variable patterns to obtain approximate CFDs. Moreover, during the traversal of $\mathcal{L}_{\text{FD}}^{\text{IT-F}}$ the equivalence partition of each item is computed on $\mathcal{D}^I$, the dataset $\mathcal{D}$ projected on I, i.e., using only those objects with a tid in $\mathsf{cov}(I, \mathcal{D})$. The algorithm FIND-FDS is then invoked on line 10, which can be any FD-discovery algorithm using equivalence partitions, to discover all FDs with confidence $\geqslant 1-\varepsilon$ on $\mathcal{D}^I$. The resulting FDs are augmented with the pattern I, and added to the set Σ of CFDs. Note that FIND-FDS is oblivious to the support threshold δ, since an FD is supported by all objects in $\mathcal{D}^I$, and $|\mathcal{D}^I| \geqslant \delta$ is already ensured by enforcing the support threshold on I.

Pseudocode for the FIND-FDS algorithm, is shown in Algorithm 3. The algorithm takes as input the first level in the lattice $\mathcal{L}_{\text{FD}}^{\text{IT-F}}$, the constant pattern P that is currently being processed by MINE-ITEMSET-FIRST, the projected database $\mathcal{D}^P$, and the confidence threshold ε. The lattice contains all variable items, except those with an attribute in $\mathsf{attrs}(P)$, with their equivalence partitions computed on $\mathcal{D}^P$. As in the other algorithms, a fringe is initialized using an arbitrary search strategy, and then traversed. For each FD $I \setminus \{j\} \to j$ encountered, the confidence of the FD $I \setminus \{j\} \to j$ on $\mathcal{D}^P$ is validated. If the FD is found to be confident, the corresponding CFD $P \cup (I \setminus \{j\}) \to j$, joined with the pattern, is added to the result.

Example 2. In the Tennis example, the itemset step of the algorithm will, for instance, visit the item $(\mathsf{Windy}, \mathsf{false})$, with $\mathsf{cov}((\mathsf{Windy}, \mathsf{false}), \mathcal{D}) = \{1,3,4,5,8,9,10,13\}$. Subsequently, an FD search is performed using only those tids in $\mathsf{cov}((\mathsf{Windy}, \mathsf{false}), \mathcal{D})$. Hence, within the FD search, the fringe is initialized with all variable items except for $(\mathsf{Windy}, \_)$, and the equivalence partitions of these single items are computed only over the tids $\{1,3,4,5,8,9,10,13\}$. The FD $(\mathsf{Outlook}, \_) \to (\mathsf{Play}, \_)$ is then found to hold, with sufficient confidence, and the CFD $\{(\mathsf{Windy}, \mathsf{false}), (\mathsf{Outlook}, \_)\} \to (\mathsf{Play}, \_)$ is added to the result. After exhausting the FD lattice for $(\mathsf{Windy}, \mathsf{false})$, the itemset mining step is resumed. ♢

Similar to the integrated approach, the final step when visiting an itemset I is to insert its children into the fringe, if they are sufficiently frequent. The

Algorithm 2 Itemset-First CFD discovery algorithm

1: **procedure** MINE-ITEMSET-FIRST($\mathcal{D}, \delta, \varepsilon$)
2: $\mathcal{L}^{\text{IT-F}} \leftarrow \{(\mathsf{A}, v) \mid \mathsf{A} \in \mathcal{A}, v \in \mathsf{dom}(\mathsf{A}), \mathsf{supp}((\mathsf{A}, v), \mathcal{D}) \geqslant \delta\}$
3: Compute $\mathsf{cov}(\{i\}, \mathcal{D})$ for all $i \in \mathcal{L}^{\text{IT-F}}$
4: Initialize *fringe* with $\mathcal{L}^{\text{IT-F}}$ depending on search strategy
5: $\Sigma \leftarrow \emptyset$
6: **while** *fringe* **not** empty **do**
7: $I \leftarrow$ POP(*fringe*)
8: $\mathcal{L}^{\text{IT-F}}_{\text{FD}} \leftarrow \{(\mathsf{A}, \_) \mid \mathsf{A} \in \mathcal{A} \setminus \mathsf{attrs}(I)\}$
9: Compute $\Pi(\{k\}, \mathcal{D}^I)$ for all $k \in \mathcal{L}^{\text{IT-F}}_{\text{FD}}$
10: $\Sigma \leftarrow \Sigma \cup$ FIND-FDS($\mathcal{L}^{\text{IT-F}}_{\text{FD}}, I, \mathcal{D}^I, \varepsilon$)
11: insert children of I into *fringe* if their support $\geqslant \delta$
12: **return** Σ

Algorithm 3 FD-discovery subroutine for Itemset-First algorithm

1: **procedure** FIND-FDS($\mathcal{L}^{\text{IT-F}}_{\text{FD}}, P, \mathcal{D}^P, \varepsilon$)
2: Initialize *fringe* with $\mathcal{L}^{\text{IT-F}}_{\text{FD}}$ depending on search strategy
3: $\Sigma \leftarrow \emptyset$
4: **while** *fringe* **not** empty **do**
5: $I \leftarrow$ POP(*fringe*)
6: **for all** $j \in I$ **do**
7: CFD $\leftarrow (P \cup I \setminus \{j\}) \rightarrow j$
8: **if** $\mathsf{conf}_{\mathsf{FD}}(\text{CFD}, \mathcal{D}^P) \geqslant 1 - \varepsilon$ **then**
9: $\Sigma \leftarrow \Sigma \cup \{\text{CFD}\}$
10: insert children of I into *fringe*
11: **return** Σ

only difference, similar to the initialization of $\mathcal{L}$, is that we again only consider constant items, with equivalence partitions boiling down to the cover of the items. The cover of each child itemset M can then be computed using a straightforward intersection of $\mathsf{cov}(I, \mathcal{D})$ and $\mathsf{cov}(J, \mathcal{D})$, for the itemsets J in the fringe with $|J| = |I|$, and such that J and I differ in only one item.

4.3.3 FD-First Discovery

The third and final approach to CFD discovery, MINE-FD-FIRST, is shown in pseudocode in Algorithm 4. This approach is a generalization of the Find-CFD algorithm [29], which starts with FD discovery. The search lattice $\mathcal{L}^{\text{FD-F}}$ is thus initialized (line 2) using only variable items:

Definition 8 (FD-First Search Lattice)**.** Consider a database instance $\mathcal{D}$. The search lattice $\mathcal{L}^{\text{FD-F}}$ for the FD-First methodology consists of all itemsets I in the powerset $\mathcal{P}(\{(\mathsf{A}, \text{`\_'}) \mid \mathsf{A} \in \mathcal{A}\})$. □

As before, equivalence partitions are computed at the beginning of the algorithm, after which a fringe is created and a breadth-first or depth-first

traversal of the lattice follows.

For every item I in the lattice, we now consider all FDs of the form $I \setminus \{j\} \to j$ for $j \in I$ (line 8). If the FD is found to be sufficiently confident, it is added to the result Σ. However, if the FD does not fully hold on the data, we additionally run an itemset mining algorithm to find all constant patterns for which the FD is sufficiently confident. During this itemset mining step, the lattice $\mathcal{L}_{\text{PAT}}^{\text{FD-F}}$ is explored, containing all constant patterns over the attributes in $\mathsf{attrs}(I \setminus \{j\})$. The first level of this lattice is initialized on line 12.

Definition 9 (FD-First Sublattice for pattern search)**.** Consider a database instance $\mathcal{D}$ and an FD $\varphi : I \setminus \{j\} \to j$ being processed in the FD-First methodology. The sublattice $\mathcal{L}_{\text{PAT}}^{\text{FD-F}}$ for the pattern search on the FD φ consists of all itemsets J in the powerset $\mathcal{P}(\{(\mathsf{A}, v) \in \mathcal{I} \mid \mathsf{A} \in \mathsf{attrs}(I \setminus \{j\})$, such that J contains at most one item per attribute, i.e., $|\mathsf{attrs}(J)| = |J|$. □

The key to the MINE-FD-FIRST method's efficiency is that the support and confidence of a considered CFD $I \setminus \{j\} \to j$ can be computed based on the information contained in $\Pi(I)$. Indeed, each equivalence class $\mathsf{eq} \in \Pi(I)$ corresponds to a unique constant pattern over the attributes $\mathsf{attrs}(I)$. By assigning a unique identifier to each class, we define the cover of an item(set) J w.r.t. the equivalence partition of I, denoted as $\mathsf{cov}(J, \Pi(I))$, as the set of identifiers of equivalence classes in which the item occurs. We call such a cover a *pidlist* (for partition id). Since typically $|\mathsf{cov}(J, \Pi(I))| \ll |\mathsf{cov}(J, \mathcal{D})|$, efficiency is increased.

Example 3. Consider the FD $\{(\mathsf{Windy}, \_), (\mathsf{Outlook}, \_)\} \to (\mathsf{Play}, \_)$ corresponding to the itemset $I = \{(\mathsf{Windy}, \_), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\}$, with equivalence class $\Pi(I) = \{\{1,8\}, \{2\}, \{3,13\}, \{4,5,10\}, \{6,14\}, \{7,12\}, \{9\}, \{11\}\}$. We assign pids to the partitions sequentially, starting from 1, e.g., $pid(\{1,8\}) = 1$ and $pid(\{11\}) = 8$. The item $(\mathsf{Windy}, \mathsf{false})$ occurs in the partitions $\{1,8\}, \{3,13\}, \{4,5,10\}$, and $\{9\}$, and hence we can now represent the constant pattern $(\mathsf{Windy}, \mathsf{false})$ by its pidlist:

$$\mathsf{cov}((\mathsf{Windy}, \mathsf{false}), \Pi(I)) = \{1,3,4,7\}.$$

Since $\mathsf{supp}((\mathsf{Windy}, \mathsf{false}), \mathcal{D}) = 8$, we have reduced the size of its cover by half. ♢

The subprocedure MINE-PATTERNS now starts by initializing a fringe containing all frequent single (constant) items over the attributes in $I \setminus \{j\}$. For each item, its pidlist has been computed from $\Pi(I)$ (line 13). Procedure MINE-PATTERNS then traverses the constant itemset lattice, generating the pidlists of new itemsets by intersecting the pidlists of two of their parents in the lattice. The support of an itemset M can be easily computed from its pidlist as follows,

$$\mathsf{supp}(M, \Pi(I)) = \sum_{pid \in \mathsf{cov}(M, \Pi(I))} |\Pi(I)[pid]|,$$

where $\Pi(I)[pid]$ denotes the equivalence class with identifier pid. Only itemsets M with $\mathsf{supp}(M, \Pi(I)) \geqslant \delta$ are considered as possible patterns for a CFD.

Algorithm 4 FD-First CFD discovery algorithm

```
1: procedure MINE-FD-FIRST(D, δ, ε)
2:     L^FD-F ← {(A, _) | A ∈ A}
3:     Compute Π({i}, D) for all i ∈ L^FD-F
4:     Initialize fringe with L^FD-F depending on search strategy
5:     Σ ← ∅
6:     while fringe not empty do
7:         I ← POP(fringe)
8:         for all j ∈ I do
9:             if conf_FD(I \ {j} → j, D) ⩾ 1 − ε then
10:                Σ ← Σ ∪ {I \ {j} → j}
11:            if conf_FD(I \ {j} → j, D) < 1 then
12:                L_PAT^FD-F ← {(A, v) | A ∈ attrs(I), v ∈ dom(A)}
13:                Compute cov({i}, Π(I)) for all i ∈ L_PAT^FD-F
14:                Σ ← Σ ∪ MINE-PATTERNS(L_PAT^FD-F, I \ {j} → j, Π(I), δ, ε)
15:        insert children of I into fringe
16:    return Σ
```

In order to form a CFD from the FD $(I \setminus \{j\}) \to j$ and the constant pattern M, where the itemset in I and M contain overlapping attributes, we need to take a set union which *replaces* those variable items in $(I \setminus \{j\})$ which have a constant counterpart in M. For this purpose, we define operator $\oplus$ as the *union with constant replacement*:

Definition 10 (Union with constant replacement ($\oplus$))**.** Consider two itemset I and J, with J consisting only of items without variable. We define

$$I \oplus J = J \cup \{(\mathsf{A}, v) \in I \mid v \neq \text{`\_'} \textbf{ or } \mathsf{A} \notin \mathsf{attrs}(J)\}. \qquad \square$$

Whenever an itemset M is processed in MINE-PATTERNS, we thus validate the CFD $(I \setminus \{j\}) \oplus M \to j$. If this CFD is sufficiently confident, it is added to the result.

Pseudocode for the MINE-PATTERNS algorithm is shown in Algorithm 5. The algorithm takes as input the first level of the lattice $\mathcal{L}_{\text{PAT}}^{\text{FD-F}}$, an FD $I \setminus \{j\} \to j$ which does not fully hold on the data, the equivalence partition $\Pi(I)$ of the itemset I containing the CFD, and the thresholds δ and ε. Recall that this lattice consists of all itemsets containing only items with attributes in $\mathsf{attrs}(I \setminus \{j\})$.

Instead of equivalence partitions or tidlists, we compute for each item the corresponding *pidlist*, i.e., its cover computed over the equivalence classes in $\Pi(I)$. Next, a fringe is initialized and traversed using an arbitrary search strategy. For every constant pattern M processed during the lattice traversal, the confidence of the CFD $(I \setminus \{j\}) \oplus M \to j$ is verified (the variable items in $I \setminus \{j\}$ are replaced by $\oplus$ with the constant items in M that have the same attribute, if such an item exists). If the CFD is sufficiently confident, it is added to the result. Both confidence and support are computed using the *pidlists* over $\Pi(I)$.

Algorithm 5 Pattern mining subroutine for FD-First algorithm

```
1: procedure MINE-PATTERNS(L_PAT^FD-F, I \ {j} → j, Π(I), δ, ε)
2:     Initialize fringe with L_PAT^FD-F depending on search strategy
3:     Σ ← ∅
4:     while fringe not empty do
5:         M ← POP(fringe)
6:         CFD ← (I \ {j}) ⊕ M → j
7:         if conf_FD(CFD, Π(I)) ⩾ 1 − ε then
8:             Σ ← Σ ∪ {CFD}
9:         insert children of M into fringe if their support ⩾ δ
10:    return Σ
```

As before, any itemset mining algorithm based on tidlists and any search strategy can be employed by MINE-PATTERNS. After the itemset mining step has finished, MINE-FD-FIRST continues by processing the remaining FDs in I, of the form $(I \setminus \{l\} \to l)$ with $l \neq j$, one by one. Finally, after all FDs in I have been processed, the children of I are added to the fringe. Since MINE-FD-FIRST only considers FDs at this level, a support check is not necessary.

We remark that the algorithm FindCFD [29] takes a similar approach, but, to our knowledge, does not perform an exhaustive search through the pattern lattice, i.e., the entire power set lattice $\mathcal{L}_{\text{PAT}}^{\text{FD-F}}$. Indeed, if an FD does not hold, this algorithm examines the equivalence partitions to obtain a *constant* CFD, without any variable patterns. As such, FindCFD discovers only FDs and constant CFDs, whereas MINE-FD-FIRST discovers general CFDs containing variables *and* constants. The fact that FindCFD does not discover all CFDs is also noted in [28].

4.3.4 Intersecting Equivalence classes

Each of the methodologies presented above makes use of equivalence partitions, and during the lattice traversals, these equivalence partitions of nodes in the lattice are intersected to obtain equivalence partitions of their children in the lattice. The algorithm for computing the intersection of two equivalence partitions, $\Pi(I)$ and $\Pi(J)$, as presented in Huhtala et al. [60], is shown using our terminology in Algorithm 6. The algorithm works as follows, a lookup table is created mapping every item in $\Pi(J)$ to the index of its equivalence class $\mathsf{eq}' \in \Pi(J)$. Then, every equivalence class $\mathsf{eq} \in \Pi(I)$ gets "split" according to the partition $\Pi(J)$: for each of the items in eq, its equivalence class index in $\Pi(J)$ is looked up, in order to partition eq into separate classes, grouping those items which are in the same class in $\Pi(J)$. Finally, all partitions in eq are added to $\Pi(I \cup J)$, and the next $\mathsf{eq} \in \Pi(I)$ is processed.

4.3.5 Time Complexity

With our three general methodologies in place, we now discuss the time complexity of CFD discovery based on equivalence partitions and tidlists. Most

Algorithm 6 Intersection algorithm for equivalence partitions

```
1:  procedure INTERSECTION(Π(I), Π(J))
2:      pIndex ← 0
3:      Lookup ← [ _ ]
4:      for all eq′ ∈ Π(J) do
5:          for all item ∈ eq′ do
6:              Lookup[item] ← pIndex
7:          pIndex ← pIndex+1
8:      Π(I ∪ J) ← ∅
9:      pIndex ← 0
10:     for all eq ∈ Π(I) do
11:         Π_pIndex ← [ _ ]
12:         for all item ∈ eq do
13:             Π_pIndex[Lookup[item]] ← Π_pIndex[Lookup[item]] ∪ {item}
14:         pIndex ← pIndex + 1
15:         for all eq″ ∈ Π_pIndex do
16:             Π(I ∪ J) ← Π(I ∪ J) ∪ {eq″}
17:     return Π(I ∪ J)
```

of the computation concerns two operations: computing equivalence partitions (or tidlists), and validating CFDs. Both operations can be performed in $\mathcal{O}(|D|)$ time. For every element I in the lattice, the equivalence partition is computed once, and $|I|$ CFDs are validated. We simplify this as $|I|$ operations per lattice element. Given that there are $|\mathcal{A}|$ attributes in the dataset, a total of $2^{|\mathcal{A}|}$ combinations of attributes exist: at level i in the lattice, there are $\binom{|\mathcal{A}|}{i}$ attribute combinations of size i. Let d denote the average size of $\mathsf{dom}(\mathsf{A})$, for $\mathsf{A} \in \mathcal{A}$. Including variable patterns, there are at most $(d+1)^i$ itemsets containing an attribute combination of size i. The number of operations performed by the algorithms is then:

$$\sum_{i=1}^{|\mathcal{A}|} \binom{|\mathcal{A}|}{i} (d+1)^i i$$

Computing this expression gives a total of $|\mathcal{A}|(d+1)(d+2)^{|\mathcal{A}|-1}$ operations, each of which is $\mathcal{O}(|D|)$. Hence, the time complexity of the algorithms is:

$$\mathcal{O}(|\mathcal{A}| \times d^{|\mathcal{A}|} \times |D|).$$

While each of our three methods performs roughly the same number of operations, the difference between them is in the time required to perform these operations. Indeed, a tidlist intersection and an equivalence partition intersection are both $\mathcal{O}(|D|)$, but in practice the tidlist intersection is faster. The Itemset-First method most efficiently computes the projected databases on which it then performs an FD-search, while the FD-First method performs much of its intersections and validation on the pidlists, which are on average

much smaller than $|D|$. These differences account for the improved performance of Itemset-First and FD-First over the Integrated approach, as experimentally shown in Section 4.5.

4.3.6 Pruning

We conclude by discussing pruning. Clearly, any CFD discovery algorithm can exploit the anti-monotonicity of support, to prune away all infrequent itemsets and their supersets. However, existing CFD discovery algorithms also provide pruning based on redundancy with respect to the antecedent of CFDs. Redundancy is defined using the concept of a preceding set:

Definition 11 (Preceding set)**.** Consider a database instance D and an itemset I containing attribute-value pairs. An itemset J is a preceding set of I, denoted $J \prec I$, if $J \neq I$ and for all $(\mathsf{A}, v) \in J$, it holds that either $(\mathsf{A}, v) \in I$, or $v =$ '_' and $(\mathsf{A}, a) \in I$, where a is a constant value in $\mathsf{dom}(\mathsf{A})$. □

Example 4. In the Tennis example, the itemsets $\{(\mathsf{Windy}, \mathsf{false}), (\mathsf{Outlook}, \_)\}$ and $\{(\mathsf{Windy}, \_), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\}$, among others, are preceding sets of the itemset $\{(\mathsf{Windy}, \mathsf{false}), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\}$. ♢

Definition 12 (CFD Redundancy)**.** Consider a database instance D and a CFD $\varphi : I \to j$ with $\mathsf{conf}(\varphi, D) \geqslant 1 - \varepsilon$. Then, φ is redundant if there exists a CFD $\varphi' : M \to n$ with $M \prec I$ and $\{n\} \preceq \{j\}$, and $\mathsf{conf}(\varphi', D) = \mathsf{conf}(\varphi, D)$. □

Example 5. In the example, CFD $(\mathsf{Temperature}, \mathrm{Cool}) \to (\mathsf{Humidity}, \mathrm{Normal})$ holds exactly. This implies the redundancy of, for example, the CFDs

$$\{(\mathsf{Temperature}, \mathrm{Cool}), (\mathsf{Humidity}, \mathrm{Normal}), (\mathsf{Windy}, \_)\} \to (\mathsf{Play}, \_)$$

$$\{(\mathsf{Temperature}, \mathrm{Cool}), (\mathsf{Windy}, \_)\} \to (\mathsf{Humidity}, \_). \quad ♢$$

Such redundancy can be eliminated efficiently in CTane (and Tane), since it employs a breadth-first traversal of the integrated search lattice, and hence all immediately preceding sets of an itemset are directly available in the level above the current one in the lattice. Pruning is then performed by associating with every itemset I in the lattice a set $\mathcal{C}^+(I)$ of candidate consequents for I and its supersets. Initially, we set $\mathcal{C}^+(I) = \{(\mathsf{A}, v) \in \mathcal{I} \mid \text{if } (\mathsf{A}, v') \in I \text{ then } v = v'\}$, i.e., all items except those for which I already contains a different item with the same attribute. Whenever a CFD is found to hold, the relevant $\mathcal{C}^+$ sets are updated, removing candidate consequents which will lead to redundant CFDs. Clearly, if $\mathcal{C}^+(I) = \emptyset$, then I and all its supersets can be removed from the search space. Updating the sets $\mathcal{C}^+$ is performed as follows in CTane:

1. If $D \models I \to j$, set $\mathcal{C}^+(M) = \mathcal{C}^+(M) \cap I$ for all M with $j \in M$ and $M \preceq I$;

2. When generating a new itemset X in the lattice, set the set of candidate consequents $\mathcal{C}^+(X) = \mathcal{C}^+(X) \cap \mathcal{C}^+(I)$ for all $I \prec X$ with $|(X \setminus I)| = 1$.

To generalize this strategy across our different approaches and search strategies, where not all preceding sets may be readily available in the search lattice, we introduce two techniques. Firstly, we use a lookup table indexed by the consequent of a rule [3], and store a list of all CFDs with that consequent that hold exactly on D. When a confident CFD $I \to j$ is found, it then suffices to verify whether a preceding set of I is present in the table at index j. If a preceding set M is found, the CFD is redundant, and pruning is performed by setting $\mathcal{C}^+(I \cup \{j\}) = \mathcal{C}^+(I \cup \{j\}) \cap M$.

Our second pruning technique is based on a generalization of the concept of free itemsets [21] (also called generators [86]). An itemset M is called free if, for all $J \subset M$, it holds that $\mathsf{supp}(J,D) \neq \mathsf{supp}(M,D)$. Moreover, it is known that all subsets of a free set are also free. We extend this concept to equivalence classes:

Definition 13 (Eq-Free Itemset)**.** An itemset I is Eq-Free in an instance D if, for all $J \subset I$, $|\Pi(I,D)| \neq |\Pi(J,D)|$ or $\|\Pi(I,D)\| \neq \|\Pi(J,D)\|$. □

We now observe that, if a CFD $\varphi : I \to j$ holds on D, then the itemset $I \cup \{j\}$ is not *Eq-Free*. Indeed, it must necessarily hold that $|\Pi(I,D)| = |\Pi((I \cup \{j\}),D)|$ and $\|\Pi(I,D)\| = \|\Pi((I \cup \{j\}),D)\|$. Hence, in order to obtain non-redundant CFDs, we additionally need to verify the Eq-Freeness of the antecedent of every considered CFD. To implement this check efficiently, we use a lookup table as in the Talky-G algorithm for mining free itemsets [101].

4.4 Related Work

Since the introduction of CFDs in Fan et al. [45], three discovery algorithms have been proposed. CTane and FastCFD were introduced in [46], and Chiang and Miller [29] presents an unnamed algorithm, which we will call FindCFD. It is important to note that FastCFD does not readily lend itself to the discovery of *approximate* CFDs, and is hence less relevant to our work. Other work on CFD discovery, such as Diallo et al. [40] and Li et al. [71], only consider constant CFDs, as opposed to general CFDs with possible variable patterns. CFD discovery can also be viewed as the discovery of a special class of conjunctive queries, as in Goethals et al. [54], but at the cost of a more time-consuming discovery process.

4.4.1 Functional Dependencies

Each of the three CFD discovery methods discussed above is rooted in the discovery of regular functional dependencies. An overview and experimental evaluation of various functional dependency discovery algorithms is presented in Papenbrock et al. [85]. The three general approaches to CFD discovery presented in this chapter can incorporate any FD discovery method making use of equivalence partitions, e.g., Tane [60], FUN [82], FD_Mine [112], and DFD [1]. Such methods support the discovery of *approximate* dependencies, and are well suited for integration with pattern

[3] We store constant CFDs $I \to (\mathsf{A}, v)$ both at indices (A, v) and $(\mathsf{A}, \_)$.

mining, due to the close relation between equivalence partitions and tidlists. We have based our implementations on the Tane algorithm, which was shown in Papenbrock et al. [85] to be the fastest algorithm on a considerable range of data sizes. The most recent work on FD discovery, Kruse and Naumann [69], makes use of a combination of equivalence partitions (called *position list indexes* (PLIs) in their paper), and so-called agree sets as in FastCFD, to improve the efficiency of the method. However, equivalence partitions are still required to compute the confidence of their approximate FDs, and hence it can be integrated into our methods.

The discovery of FDs has also been addressed from the perspective of Formal Concept Analysis (FCA). Lopes et al. [73] presents an SQL based algorithm for FD discovery, and Baixeries et al. [10] presents a method based on pattern structures, but does not discuss the discovery of approximate FDs. The link between FCA and CFDs is characterized theoretically in Medina and Nourine [80]. For an overview of FCA, we refer to Ganter and Wille [50].

Although interestingness measures for FDs based on statistical tests have been proposed in Mandros et al. [75], we consider approximate CFDs defined in terms of support and confidence as these are most widely used in the data quality context. In a similar vein, Berti-Équille et al. [15] discovers FDs over data with missing values based on a measure of *genuineness*, which corresponds to the likeliness that the FD would hold after imputation of the missing values. In contrast, we treat values such as *NULL* or *"?"* simply as another value in the domain, which is called the `NULL-EQ` strategy in Berti-Équille et al. [15]. Such alternative interestingness measures could optionally be integrated into our methods, but are orthogonal to the content of this chapter.

4.4.2 Association Rules

Association rules (ARs) were first introduced in Agrawal et al. [5] for supermarket basket analysis. The discovery of ARs is based on mining frequent patterns, which started with the APriori algorithm in Agrawal et al. [6], and has received much attention since. Of particular interest to our approaches for CFD discovery are so-called vertical itemset mining algorithms, which employ a vertical data layout for efficient frequency computation, such as Eclat [116]. Such algorithms are well-suited for integration with FD discovery, since the vertical data layout relates naturally to the equivalence partitions used in FD discovery. Our implementations are based on Eclat. Other important algorithms for AR mining include FP-Growth, presented in Han et al. [57], and Opus, presented in Webb [107]. For overviews of itemset and association rule mining, we refer to Goethals [53] and Zaki and Meira Jr [115].

Throughout this chapter, we have viewed CFDs as a kind of ARs. An in-depth discussion of the relationships between FDs, CFDs, and ARs can be found in Medina and Nourine [79].

Table 4.1: Statistics of the UCI datasets used in the experiments. We report the number of tuples, distinct constant items, and attributes.

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$
Adult	48842	202	11
CreditCard	30000	216	12
Mushroom	8124	119	15
Nursery	12960	32	9

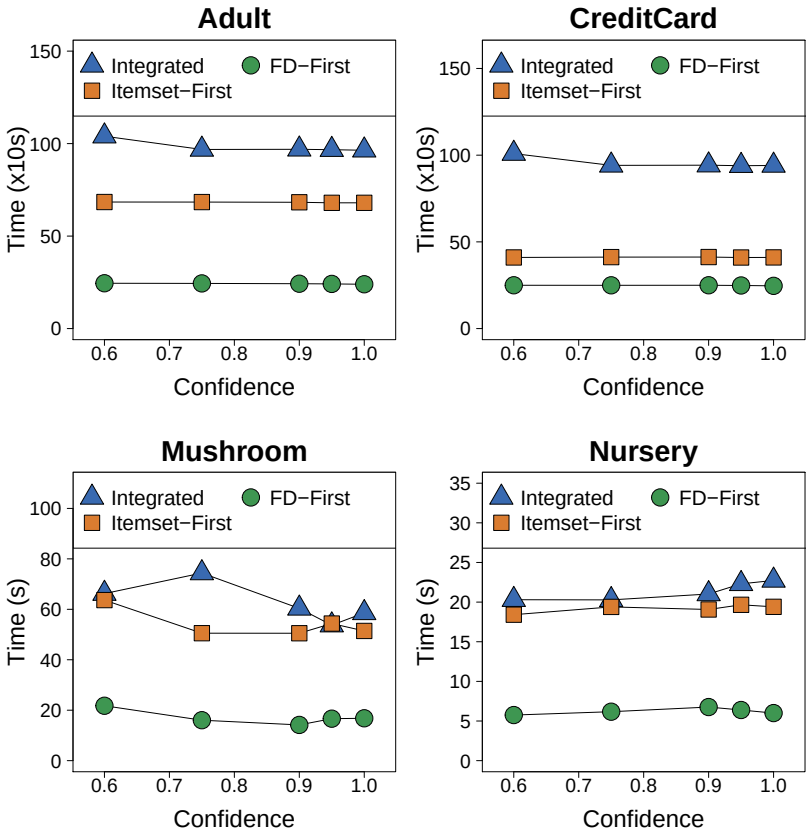


Figure 4.2: Influence of confidence threshold on runtime of three CFD discovery algorithms.

4.5 Experiments

We experimentally validate the proposed techniques on real-life datasets from the UCI repository [39], described in Table 4.1. The mushroom dataset was restricted to its first 15 attributes, as runtimes became too high when considering more attributes. The algorithms have been implemented in C++,

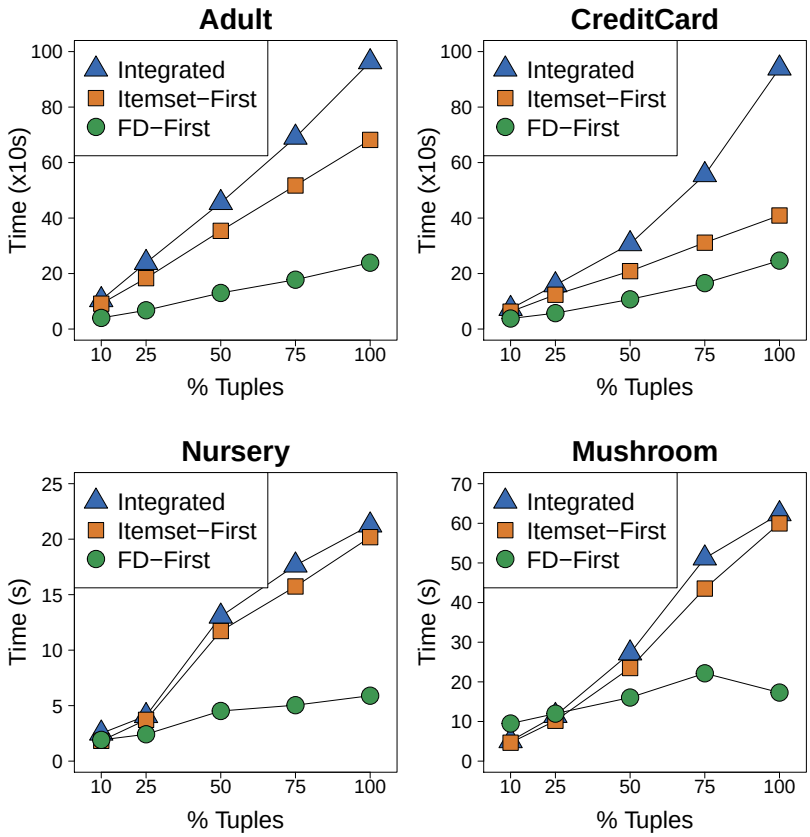


Figure 4.3: Scalability of three CFD discovery algorithms in number of tuples.

the source code and used datasets are available for research purposes [90]. The program was tested on an Intel Xeon Processor (3.8GHZ) with 32GB of memory running Ubuntu. Our algorithms run entirely in main memory. All runtimes were obtained as an average over 3 independent runs.

In Section 4.3, we have described the three approaches to CFD discovery in full generality, i.e., using any FD discovery algorithm based on equivalence partitions, any itemset mining algorithm using tidlists, and any search strategy. We begin the experimental section by describing specific instantiations of our approaches, which were used in the experiments:

Integrated uses a depth-first implementation of the CTane algorithm

Itemset-First uses a breadth-first version of Eclat for the itemset mining step, and a depth-first Tane implementation for the FD discovery step

FD-First uses both a depth-first Tane step and depth-first itemset mining

All our depth-first implementations use a reverse pre-order traversal. We selected these three instantiations as the *best ones* – in terms of efficiency –

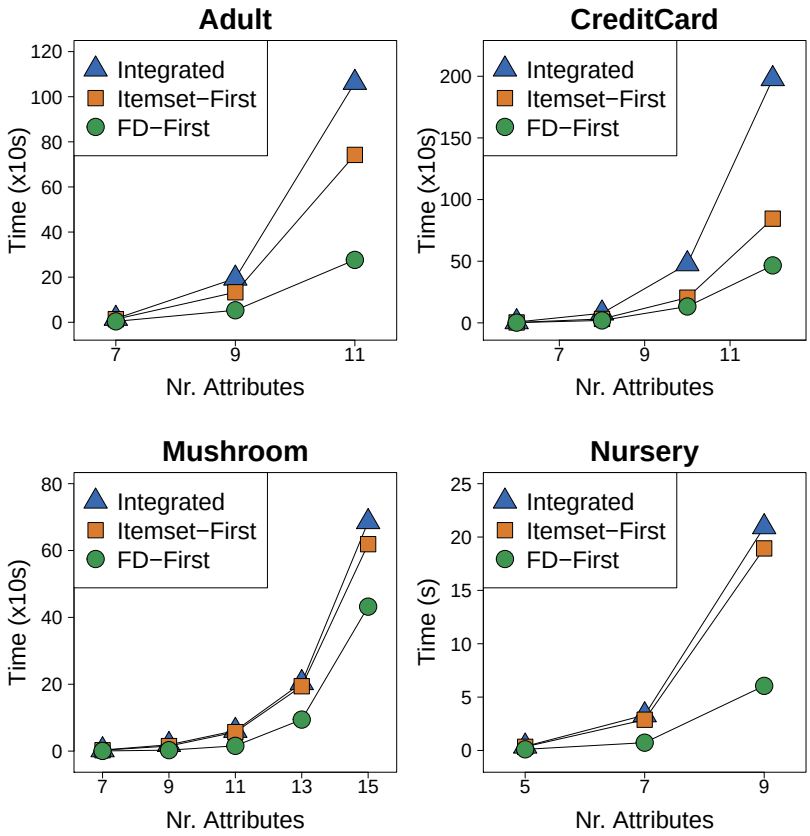


Figure 4.4: Scalability of three CFD discovery algorithms in number of attributes.

out of a total of 18 different combinations. The runtime results of all instantiations are available in appendix A.1. The above choices did not impact the relative ordering between the different methodologies. Since the Integrated approach is similar to the CTane algorithm, we consider this as the benchmark, i.e., state-of-the-art in CFD discovery.

Since CFD (and FD) discovery is inherently exponential in the number of attributes of a dataset, we sometimes reduce the overall runtimes of the algorithms by enforcing a limit on the size of rules, called the maximum antecedent size. We compare the runtime of the three methodologies in function of the number of tuples and attributes of the data, the minimum support threshold, and the maximum antecedent size. We emphasize that all methods return the exact same result in every experiment.

4.5.1 Confidence Threshold

First, we investige the influence of the confidence threshold on the runtime of the algorithms. Runtime results in function of confidence are shown in

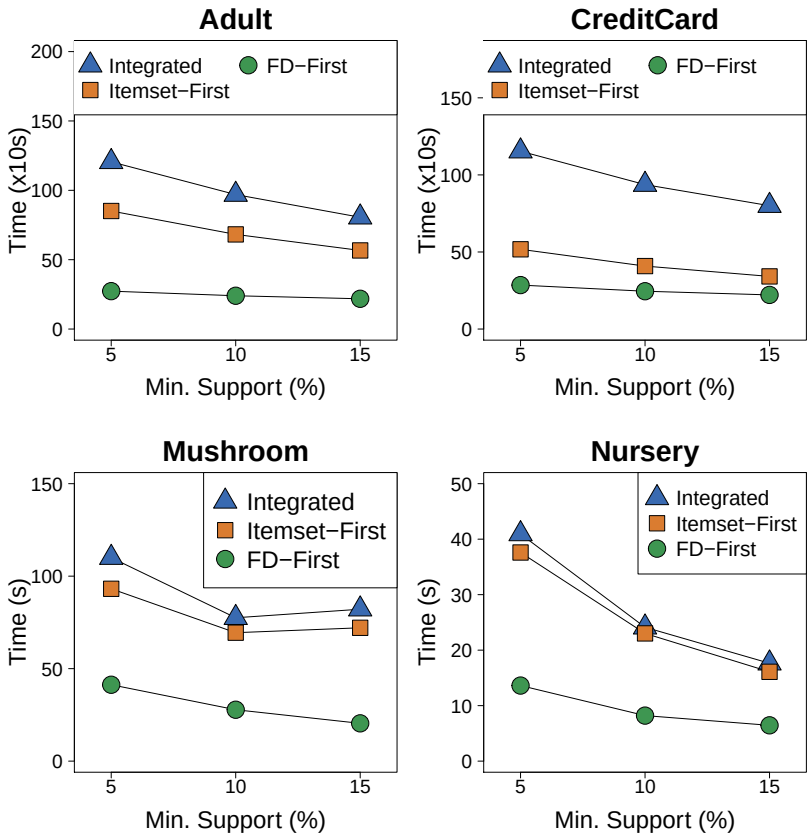


Figure 4.5: Scalability of three CFD discovery algorithms in minimum support threshold.

Figure 4.2. These results were obtained with a minimum support of 10%, and a maximum antecedent size of 6. As stated at the beginning of the experimental section, the confidence threshold has a negligible impact the runtime of CFD discovery. This makes sense: since no pruning occurs based on confidence, all these CFDs are validated regardless of the threshold, and the only difference is whether they are added to the result. As a result, we only mine exact CFDs in the remainder of this experimental section.

4.5.2 Number of Tuples

We next investigate the scalability of each approach in terms of the number of tuples. For this experiment, we take each dataset, and consider only the first $X\%$ tuples. We consider the values $10\%, 25\%, 50\%, 75\%$, and 100% for X. The minimum support threshold was fixed at 10% of the number of tuples considered, and thus grows as the size of the considered subset of the database increases. The maximum antecedent size was fixed at 6. The obtained runtimes are displayed in Figure 4.3. The three methodologies exhibit

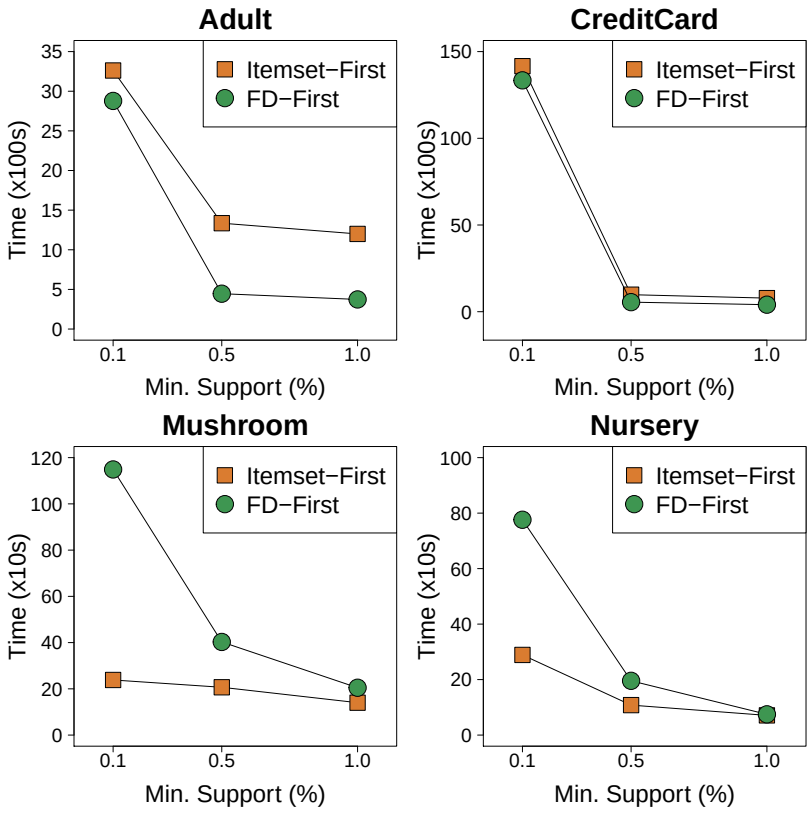


Figure 4.6: Scalability of Itemset-First and FD-First discovery algorithms for very low minimum support thresholds.

a linear increase in runtime as the number of tuples increases. We see that the FD-First approach scales better than the other approaches, and is faster overall, followed by the Itemset-First approach, and then the Integrated approach.

4.5.3 Number of Attributes

Similar to the previous experiment, we now investigate the performance of the three algorithms in terms of the number of attributes in the dataset. We consider only the first X attributes, with X increasing in increments of two. In Figure 4.4, the runtimes are shown on each dataset for increasing values of X. The minimum support threshold and maximum antecedent size were again fixed at 10% and 6, respectively. As expected, the CFD discovery problem is inherently exponential in the number of attributes. This rise in runtime is clearly visible for each of the three methodologies. However, while all methods are exponential, the FD-First method clearly outperforms the other approaches. The Integrated method is the slowest overall, and suffers most

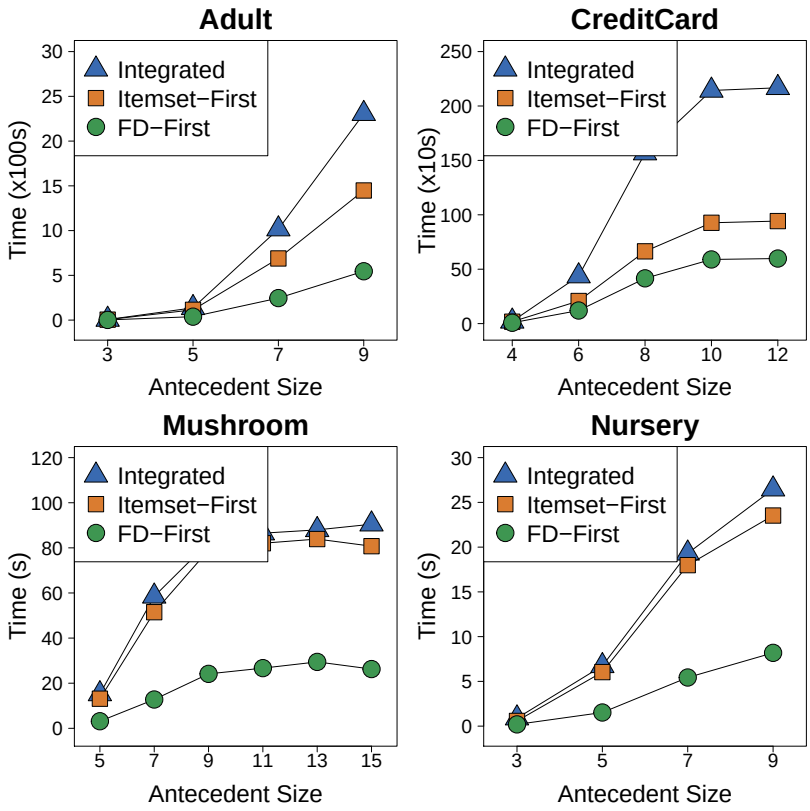


Figure 4.7: Scalability of three CFD discovery algorithms in maximal size of antecedent.

of all from the increasing number of attributes.

4.5.4 Minimum Support

We next fix the dimensionality of the data, using all tuples and attributes, and study the influence of the minimum support threshold on runtime. The results for the three datasets are shown in Figure 4.5, for minimum support thresholds of 5%, 10%, and 15% of the total number of tuples. Overall, we see that the runtime decreases slightly as the support threshold increases. Clearly, support has less impact on the runtime of the methods than the number of tuples and attributes. The FD-First method shows the lowest increase in runtime as support decreases, and is clearly the fastest method, while the other two methods show a similar increase for lower supports.

However, this situation changes when considering very low support thresholds. In Figure 4.6, we show runtimes for the Itemset-First and FD-First methods for minimum supports ranging of 0.1%, 0.5%, and 1%. We do not display the Integrated approach, since it is much slower in this sup-

Table 4.2: Number of (approximate) cfds discovered for various support and confidence thresholds.

Dataset	Minsup	Conf = 1.0	Conf = 0.99	Conf = 0.95
Adult	15%	3	8028	11426
	10%	7	11841	19342
	5%	32	22256	44962
CreditCard	15%	0	1970	21802
	10%	5	3429	32266
	5%	24	8711	62244
Mushroom	15%	3739	219344	877583
	10%	5842	314259	1240933
	5%	11117	438325	1969904
Nursery	15%	2	2	80
	10%	7	7	225
	5%	25	57	836

port range, distorting the plot. As support becomes very low, the FD-First method shows a strong increase in runtime, whereas the Itemset-First method is much less impacted. Indeed, for such low supports, the pattern mining step becomes the most expensive part of CFD discovery, which is handled most efficiently by the Itemset-First approach. This clearly indicates that both methodologies have their merit, in different contexts.

4.5.5 Maximal Antecedent Size

We conclude our runtime experiments by investigating the impact of the maximal antecedent size threshold on the runtime of the algorithms. Recall that this threshold limits the size of the discovered CFDs, and thus, the depth of the search tree. The minimum support threshold was again fixed at 10%. The results are shown in Figure 4.7. We see an exponential increase in runtime, similar to that observed when the number of attributes was increased. This makes sense, since both parameters directly correspond to the depth of the search tree. The FD-First approach again performs best on every dataset, and shows the lowest increase in runtime as antecedent size increases. As before, the Itemset-First approach is faster than the Integrated approach.

4.5.6 Number of CFDs

As additional information, we show the number of CFDs found for various support and confidence thresholds in Table 4.2. We only use very high confidence thresholds, as such CFDs are typically used for data cleaning. The

number of CFDs increases quickly as the number of attributes increases, as is the case on the Mushroom dataset. Moreover, while the number of CFDs is manageable for high confidence thresholds, CFD discovery also suffers from pattern explosion when considering low confidence thresholds. As noted in the beginning of this experimental section, all algorithms returned the exact same set CFDs.

4.6 Conclusion

In this chapter, we have presented the discovery of Conditional functional dependencies (CFDs) as a form of association rule mining, and classified the possible discovery approaches into three categories. These categories are based on how the different approaches combine pattern mining and functional dependency discovery. Two of these approaches have not been considered before. Moreover, we discuss how CFD discovery and pruning can be performed independent of methodology and search strategy, either breadth-first or depth-first.

We show experimentally that both our new approaches, FD-First and Itemset-First, outperform the baseline Integrated method, which was based on the existing CTane algorithm. In most cases, the FD-First methodology performs best, and is considerably faster than the baseline. We further identify situations in which either FD-First or Itemset-First achieves the best performance, indicating that both methodologies have individual merit.

Most crucially, we have shown that the field of CFD discovery still offers opportunities for improvement. This is highly relevant in view of the popularity of CFDs in data cleaning. While CFDs are frequently used for data cleaning, their efficient discovery has not received much attention in recent years.

CHAPTER 5

Explaining Repaired Data with CFDs

5.1 Introduction

After addressing the discovery of constraints, in the form of CFDs, we now turn our attention towards user involvement in the discovery of CFDs. Typically, only a human user can provide semantic knowledge of the data. Such semantic knowledge is necessary in order to discover constraints that are *semantically valid* and *useful for repairing*.

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*. Since such repairs encode both which values are erroneous, and how they should be corrected, they provide much useful information in a compact format. 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 information provided by high-precision error detection and data profiling methods [95, 14, 88]: 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

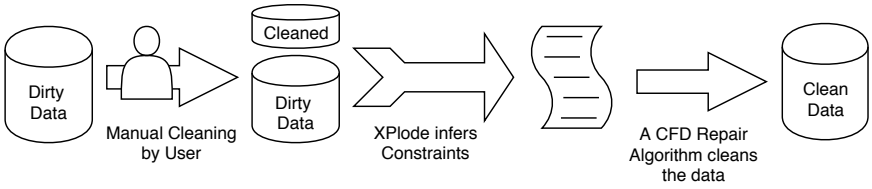


Figure 5.1: Schematic overview of the proposed XPlode workflow.

performed using any state-of-the-art repair algorithm, such as those presented in Bohannon et al. [18], Dallachiesa et al. [38], Galhardas et al. [49], Geerts et al. [52], Khayyat et al. [65], Kolahi and Lakshmanan [67], among others. This proposed workflow is shown schematically in Figure 5.1

In this chapter, 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.

5.1.1 Methodology

As the underlying constraint formalism for our method, we use the class of *conditional functional dependencies* (CFDs) [45]. Our proposed method, called XPLODE (for e**Xpl**anation **o**n **de**mand), extracts a single constraint, the *best explanation*, from a dirty dataset and an associated repair. To assess the quality of an explanation, we introduce a scoring function that quantifies how much of the repair is explained by a CFD. We further prove that the outcome of XPLODE is equivalent to a "naive" method that first discovers all CFDs, and then uses a post-processing step to find the explanation with maximal score.

Our algorithm is "on-demand" in that it dynamically generates candidate explanations, only if they can potentially improve on the current best explanation, hereby attempting to quickly discover the best possible explanation of the partial repair. A similar on-demand exploration was used in Golab et al. [55] 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 [19]. This class includes useful functions such as a minimum, maximum, or average, and our scoring function. Moreover, we adapt the algorithm to our specific setting, implementing various optimizations.

Underlying the XPLODE algorithm 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 [58, 102] is that (i) we *only require modifications* as user input,

Table 5.1: 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	EH41DT
6	44	131	4444444	Ian	High St.	EDI	EH41DT
7	44	908	4444444	Ian	Port PI	MH	W1B1JH
8	~~44~~ 01	131	2222222	Sean	3rd Str.	UN	01202

and (ii) we consider *general CFDs*. In contrast to our method, He et al. [58] uses a *single* modification to bootstrap a *constant* CFD discovery process. This process then proceeds by asking the user to verify the validity of the explored *constraints*. Similarly, Thirumuruganathan et al. [102] finds FDs that capture errors by post-processing a set of approximate FDs discovered up front. The post-processing step again involves questions to the user about the validity of data and *constraints*. The question that remains is, how should the user assess the validity of a presented rule? As argued above, in the worst case, this involves checking all the tuples that match the (C)FD in question. And what happens if the user makes a wrong assessment?

Instead of frequently requiring a user to *invalidate* a candidate constraint, our method only requires *positive* feedback in the form of correct modifications. 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.

5.1.2 Motivating Example

For illustration, we borrow the running example from Fan et al. [46]. Table 5.1 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[CT] from "LA" to "MH" and in t_2[CT] from "GLA" to "MH", and the country code (CC) in t_8[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, many other CFDs become candidate explanations, such as $([\mathsf{CC},\mathsf{PN}] \rightarrow \mathsf{CT}, (01, 1111111, \mathrm{MH}))$, and $(\mathsf{NM} \rightarrow \mathsf{CT}, (\mathrm{Mike}, \mathrm{MH}))$. 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. Instead, 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 one of the many candidate explanations can be related to the two modifications made by the user: the FD $\varphi = ([\mathsf{AC},\mathsf{CC}] \rightarrow \mathsf{CT})$, saying that country code and area code (AC) uniquely determine city, which is known to be a semantically valid constraint on this data. Moreover, φ can now be used to automatically correct $t_2[\mathsf{CT}]$ to "MH", using any CFD-based repair algorithm.

5.1.3 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 5.2)

2. We design an algorithm, XPLODE, that discovers the best explanation, i.e., the explanation of highest score. Moreover, the XPLODE algorithm is *on-demand* in the sense that it 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 5.3)

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)

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 5.5)

5.2 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 5.2.1. How modifications can be *explained* in terms of CFDs, and how to differentiate between different explanations based on some *scoring function*, is discussed in Section 5.2.2. Our formal problem statement is given in Section 5.2.3. Throughout this chapter, we consider only tabular datasets.

5.2.1 Modifications

We consider a setting in which no CFDs are provided alongside the (dirty) database instance. Instead, we have at our disposal *two* tabular 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 modified tuple in D_{rep} contains one or more modifications, encoding the changes made to this tuple by the user:

Definition 14 (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 combination of 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 afflicted by the set of modifications M, i.e., those tuples whose tids occur in a modification in M. Furthermore, we let $\sigma_M^{\mathsf{tid}}(D)$ denote the set of tids in $\sigma_M(D)$.

Example 6. In our example, the set $\mathfrak{M}$ consists of three modifications:

$$\mathfrak{m}_1 = (1, \mathsf{CT}, \mathrm{LA}, \mathrm{MH}),\ \mathfrak{m}_2 = (2, \mathsf{CT}, \mathrm{GLA}, \mathrm{MH}),\ \mathfrak{m}_3 = (8, \mathsf{CC}, 44, 01).$$

Both $\sigma_{\mathfrak{M}}(D_{dirty})$ and $\sigma_{\mathfrak{M}}(D_{rep})$ consist of the tuples t_1, t_2 and t_8, with tids 1, 2 and 8, in D_{dirty} and D_{rep}, respectively. ♢

5.2.2 Explaining a Repair

Suppose that we are given a repair, represented by a set of modifications made by a user. We do not assume that the user has any knowledge of which CFDs may be required to hold on the data, in order to capture 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 then 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 the number of candidate CFDs, we will only consider (ε,δ)-CFDs, which are both confident and frequent in the data. We illustrate this choice with an example.

Example 7. 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 $\varphi_1 = (\mathsf{NM} \rightarrow \mathsf{CT}, (\mathrm{Mike}, \mathrm{MH}))$ 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 $\varphi_2 = (\emptyset \rightarrow \mathsf{CT}, (\mathrm{MH}))$, 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. $\diamondsuit$

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}. We are thus interested in (ε,δ)-CFDs, as defined in Section 2.1.

Definition 15 ((ε,δ)-CFDs)**.** Given a database instance D_{rep}, support threshold δ and confidence threshold ε, we denote by $\Sigma_{(\varepsilon,\delta)}(D_{rep})$ the set of all (ε,δ)-CFDs on D_{rep}. $\square$

From here on, we will use the set $\Sigma_{(\varepsilon,\delta)}(D_{rep})$ as our candidate set of CFDs for explaining repairs. The obvious question is then, 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 of a modified tuple, i.e., the intersection between the violations and $\sigma^{\mathsf{tid}}_{\mathfrak{M}}(D_{dirty})$ is not empty. 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 (but not on all of D_{rep}). 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 of modifications $M \subseteq \mathfrak{M}(D_{dirty}, D_{rep})$:

Definition 16 (M-Repair Explanation)**.** Consider instances D_{dirty}, D_{rep}, and modifications $M \subseteq \mathfrak{M}(D_{dirty}, D_{rep})$. A CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ is an *M-repair explanation* if

1. $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty} \oplus M) > \mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty})$;
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 CFDs in $\Sigma_{(\varepsilon,\delta)}(D_{dirty} \oplus M)$ which are M-repair explanations. □

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 5.5). 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 by the following example.

Example 8. When discovering global explanations in our example dataset, with $\varepsilon = 0.25$ and $\delta = 2$, there are 18 candidate CFDs. Among these candidates, the 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, in order to define the "best" explanation. Observe that in the previous example, the CFD $([\mathsf{AC},\mathsf{CC}] \rightarrow \mathsf{ZIP},(\_,\_,\_))$ 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 8 indicates that good global explanations should also explain the modifications involved *locally*. Intuitively, this means that the three conditions stated in Definition 16 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. For instance, imagine a user making a mistake in their manual repairs. We start by defining what it means for a CFD to locally explain a set of modifications.

Definition 17 (Local Repair Explanation)**.** Given a CFD φ in the set of $\mathfrak{M}$-explanations $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{rep})$, 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,

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

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 9. In the running example, the CFD $([\mathsf{CC},\mathsf{AC}] \rightarrow \mathsf{ZIP},(\_,\_,\_))$ 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 $([\mathsf{CC},\mathsf{AC}] \rightarrow \mathsf{CT},(\_,\_,\_))$ 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. ♢

5.2.3 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 [46, 29], exhibit an inherent exponential dependency in $|\mathcal{A}|d$, where $|\mathcal{A}|$ is the number of attributes and d is the average number of values in the domain of an attribute, as we have shown in Section 4.3.5. 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 5.4, that an efficient and good approximation of the scoring function can be computed.

5.3 Discovering Repair Explanations

We next describe an algorithm for discovering the best repair explanation, as defined in our problem statement. One possible 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 5.3.1 with a detailed overview of XPLODE. Correctness of XPLODE is shown in Section 5.3.2, 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 5.3.3, we briefly discuss how to modify XPLODE for returning multiple CFDs. Further implementation details on the computation of support, confidence, and checking for global explanations, are presented in Section 5.3.4.

5.3.1 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 7 for its pseudo-code. At the core of XPLODE is a traversal of a lattice, as introduced in Chapter 4. We again

base our CFD discovery algorithm on equivalence partitions, defined in Section 2.1. We elaborate further on this in Section 5.3.4, where we translate the definitions from Section 2.1, which are based on itemsets and a transaction dataset, to our current context of tabular datasets. For now, it suffices to know 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 then to quickly pinpoint a CFD in the set $\mathsf{Explain}_{(\varepsilon,\delta)}(D_{dirty} \oplus \mathfrak{M})$ with guaranteed highest score.

To explain the workings of XPLODE, we need to introduce some concepts. First of all, the algorithm is based on the traversal of the integrated search lattice defined in Chapter 4 (Definition 2). In terms of a tabular dataset, 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, (Y, s_p) is a child of (X, t_p), denoted $(\mathsf{X}, t_p) \subseteq (\mathsf{Y}, s_p)$, if and only if $\mathsf{X} \subseteq \mathsf{Y}$ and $t_p = s_p[\mathsf{X}]$. 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 the threshold 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 CFD $\varphi \in \mathsf{CandCFD}(\mathsf{X}, t_p)$. 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 set of 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.

Algorithm 7 On-demand algorithm XPLODE for obtaining the best explanation for a set $\mathfrak{M}$ of modifications.

1: **procedure** XPLODE($D_{dirty}, D_{rep}, \mathfrak{M}, \varepsilon, \delta, \mathsf{score}(\cdot), \mathsf{UB}(\cdot)$)
2: $\Phi \leftarrow$ PRORITYQUEUE($\{\}$)
3: Insert $(\emptyset,\emptyset)$ into Φ with upper bound $+\infty$
4: $\varphi_{\max} \leftarrow$ nil, max $\leftarrow 0$
5: **while** Φ is not empty **do**
6: $(\mathsf{X},t_p) \leftarrow$ POP(Φ)
7: Let $\varphi \in \mathsf{CandCFD}(\mathsf{X},t_p)$ such that φ is a global explanation with highest score among all global explanations in $\mathsf{CandCFD}(\mathsf{X},t_p)$
8: **if** φ exists and $\mathsf{score}(\varphi,\mathfrak{M}) > \mathsf{max}$ **then**
9: $\varphi_{\max} \leftarrow \varphi$, max $\leftarrow \mathsf{score}(\varphi,\mathfrak{M})$
10: Delete from Φ all elements with UB-value $\leqslant$ max
11: **for all** children (Y,s_p) of (X,t_p) **do**
12: **for all** $\psi \in \mathsf{CandCFD}(\mathsf{Y},s_p)$ **do**
13: Compute $\mathsf{score}(\psi,\mathfrak{M})$
14: **if** $\mathsf{UB}(\mathsf{Y},s_p) > \mathsf{max}$ **then**
15: Insert (Y,s_p) into Φ with value $\mathsf{UB}(\mathsf{Y},s_p)$.
16: **return** $\varphi_{\max}$.

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*.

Tie Breaking. We remark that 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.

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. In-

deed, 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.

5.3.2 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, the current highest score.

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* [19]: 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 $\mathsf{UB}(\mathsf{X},t_p) \leqslant \mathsf{UB}(\mathsf{Y},s_p)$. That is, every element in the lattice has at least one parent with a higher or equal UB-value.

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.

Proof. Let $\varphi_{\max} \neq \mathsf{nil}$ be the CFD returned by XPLODE, and let max be the score of $\varphi_{\max}$. Assume, for the sake of contradiction, that there exists a global explanation $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ with $\mathsf{max} < \mathsf{score}(\varphi, \mathfrak{M})$. Since $\mathsf{UB}(\mathsf{X},t_p) \geqslant \mathsf{score}(\varphi,\mathfrak{M}) > \mathsf{max}$, this implies that $(\mathsf{X} \cup \{\mathsf{A}\}, t_p)$ was never added to the priority queue Φ.

Note, however, that since $\mathsf{UB}(\cdot)$ is loose anti-monotonic, there exists a path in the lattice from $(\mathsf{X} \cup \{\mathsf{A}\}, t_p)$ to an attribute/value pair (B,b), at the first level of the lattice, along which the UB-values never decrease. Clearly, (B,b) is added to the priority queue when $(\emptyset,\emptyset)$ was expanded. This in turn implies that all elements on the path from (B,b) to (X,t_p), all having a UB-value greater than max, must have been added to the queue before the algorithm can terminate. This contradicts our earlier observation. We may thus indeed conclude correctness. □

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.

Example 10. We return to the running example, and consider the CFD $\varphi = (\mathsf{CC} \to \mathsf{CT}, (\_, \_))$, 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 $\varphi_1 = (\emptyset \to \mathsf{CT}, (\_))$ and $\varphi_2 = (\emptyset \to \mathsf{CC}, (\_))$ 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. $\diamondsuit$

Instead, we base our upper bound on the following observation. For a CFD φ, we define the set $\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})$ 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 17 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, it must hold that $\mathsf{score}(\varphi, \mathfrak{M}) \leqslant |\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})|$. We can now define our upper bound function UB_0.

Definition 18 (Upper bound)**.** Let (X, t_p) be a lattice element, D_{dirty} a dirty dataset, and $\mathfrak{M}$ a set of modifications.

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

It turns out that:

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

Proof. Consider an element (X, t_p) in the lattice. Let $\varphi = (\mathsf{X} \setminus \{\mathsf{A}\}, t_p)$ be the CFD in $\mathsf{CandCFD}(\mathsf{X}, t_p)$ such that $\mathsf{UB}_0(\mathsf{X}, t_p) = |\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})|$. Let $\mathsf{Y} = \mathsf{X} \setminus \{\mathsf{B}\}$ for some $\mathsf{B} \in \mathsf{X} \setminus \{\mathsf{A}\}$. We observe that for CFD $\psi = (\mathsf{Y} \setminus \{\mathsf{A}\} \to \mathsf{A}, t_p[\mathsf{Y}])$, we have that $\mathsf{VIO}(\varphi, D_{dirty}) \subseteq \mathsf{VIO}(\psi, D_{dirty})$, and hence it must also hold that $|\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})| \leqslant |\mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})|$. Furthermore, CFD ψ is in $\mathsf{CandCFD}(\mathsf{Y}, t_p[\mathsf{Y}])$. As a consequence, $\mathsf{UB}_0(\mathsf{Y}, t_p[\mathsf{Y}]) \geqslant |\mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})|$ which is larger or equal than $\mathsf{UB}_0(\mathsf{X}, t_p) = |\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})|$. It suffices to observe that $(\mathsf{Y}, t_p[\mathsf{Y}])$ is a parent of (X, t_p). $\square$

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 $\mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})$

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

as the set of attributes occurring in $\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})$. Furthermore, we let λ be a parameter such that $0 \leqslant \lambda \cdot |\mathcal{A}| < 1$, 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 "$+1$" here serves 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 $\lambda \cdot |\mathcal{A}| \leqslant 1$. Hence, $\mathsf{UB}_\lambda(\cdot)$ only affects the priority among those CFDs explaining an identical number of modifications.

Proposition 3. The upper bound function $\mathsf{UB}_\lambda(\cdot)$ satisfies the conditions of Proposition 1.

Proof. We use a similar argument as in the proof of Proposition 2, showing that $\mathsf{UB}_0(\cdot)$ is loose anti-monotonic. That is, consider element (X, t_p) and a candidate CFD $\varphi = (\mathsf{Y} \to \mathsf{A}, t_p)$ for $\mathsf{Y} = \mathsf{X} \setminus \{\mathsf{A}\}$ such that $\mathsf{UB}_\lambda(\mathsf{X}, t_p)$ is equal to

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

Let $\mathsf{B} \in \mathsf{Y}$ and CFD $\psi = (\mathsf{Y} \setminus \mathsf{B} \to \mathsf{B}, t_p[\mathsf{Y}])$. We have $\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M}) \subseteq \mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})$. We distinguish between the following cases:

(a) $\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M}) = \mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})$, in which case it follows that $\mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M}) = \mathsf{AttVIO}(\psi, D_{dirty}, \mathfrak{M})$. Since $\mathsf{Y} = \mathsf{X} \setminus \mathsf{A}$, it is trivially verified that

$$|\mathsf{X} \cup \mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})| \geqslant |(\mathsf{X} \setminus \mathsf{A}) \cup \mathsf{AttVIO}(\psi, D_{dirty}, \mathfrak{M})|,$$

and hence we have that

$$\begin{aligned}
&|\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})| + 1 - \lambda |\mathsf{X} \cup \mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})| \\
&= |\mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})| + 1 - \lambda |\mathsf{X} \cup \mathsf{AttVIO}(\psi, D_{dirty}, \mathfrak{M})| \\
&\leqslant |\mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})| + 1 - \lambda |\mathsf{Y} \cup \mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})|.
\end{aligned}$$

(b) $\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M}) \subset \mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})$, in which case it follows that $|\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})| + 1 \leqslant |\mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})|$. Furthermore, $\mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})$ can be at most $\mathcal{A}$. It remains to verify that

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

where $|\mathsf{X} \cup \mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})| \geqslant 0$. We obtain that $1 - \lambda |\mathcal{A}| \geqslant 0$, which is equivalent to our assumption that $\lambda |\mathcal{A}| \leqslant 1$. Thus, we may infer that also in this case

$$\begin{aligned}
&|\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})| + 1 - \lambda |\mathsf{X} \cup \mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})| \\
&\leqslant |\mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})| + 1 - \lambda |\mathsf{Y} \cup \mathsf{AttVIO}(\psi, D_{dirty}, \mathfrak{M})|.
\end{aligned}$$

Since ψ is a candidate CFD of $(Y, t_p[Y])$, which is a child of (X, t_p), we have that $\mathsf{UB}_\lambda(Y, t_p[Y]) \geqslant |\mathsf{ModVIO}(\psi, D_{dirty}, \mathfrak{M})| + 1 - \lambda|Y \cup \mathsf{AttVIO}(\psi, D_{dirty}, \mathfrak{M})|$, which in turn is at least $|\mathsf{ModVIO}(\varphi, D_{dirty}, \mathfrak{M})| + 1 - \lambda|X \cup \mathsf{AttVIO}(\varphi, D_{dirty}, \mathfrak{M})|$, which was chosen to coincide with $\mathsf{UB}_\lambda(X, t_p)$. Hence, $\mathsf{UB}_\lambda(\cdot)$ is indeed loose anti-monotonic. □

5.3.3 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 explanations that incrementally explain the modifications. We believe a combination of these techniques covers most real-world scenarios.

Both strategies assume that a number of modifications are consistent with respect to a single CFD; in other words, that a part of the given repair contains sufficient information to retrieve the CFD corresponding to that part of the repair. Further improvements, for instance to prevent overfitting, can easily be made and integrated into the algorithm by tweaking the scoring function.

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 $\varphi \in \mathsf{CandCFD}(X, t_p)$ 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 efficiently 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 $\mathsf{UB}(Y, s_p) > 0$, 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 $\mathsf{UB}(Y, s_p) = 0$ are

removed, and the others are used to re-initialize the priority queue Φ. The algorithm then repeats, until all modifications are explained.

5.3.4 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. We have previously discussed equivalence partitions in Section 2.1, in terms of a transaction dataset $\mathcal{D}$ and itemset I. In this section, we briefly recap the necessary concepts, and translate them into the corresponding terms of a tabular dataset.

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 $\Pi(\mathsf{X}, t_p)$. In the running example of this chapter, we then have that $\Pi(\{\mathsf{CC}, \mathsf{CT}\}, (44, \_)) = \{\{5,6\},\{7\}\}$ with $|\Pi(\{\mathsf{CC}, \mathsf{CT}\}, (44, \_))| = 2$ and $\|\Pi(\{\mathsf{CC}, \mathsf{CT}\}, (44, \_))\| = 3$.

Recall that the key use of equivalence partitions is to check the validity of CFDs, since $D_{rep} \models (\mathsf{X} \to \mathsf{A}, t_p)$ if and only if $|\Pi(\mathsf{X}, t_p[\mathsf{X}])| = |\Pi(\mathsf{X} \cup \{\mathsf{A}\}, t_p)|$. Below, we again make use of the concept of *refinement* of an equivalence partition (Definition 3): for an equivalence class $\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])$, we let $\Pi(\mathsf{X} \cup \{\mathsf{A}\}, t_p)^{\mathsf{eq}}$ 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 $|\Pi(\mathsf{X} \cup \{\mathsf{A}\}, t_p)^{\mathsf{eq}}| = 1$ for every $\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])$.

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, as in Chapter 4, 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 [60]. 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 $\mathsf{supp}(\varphi, D_{rep}) \geqslant \delta$. Recall from the

previous chapter that confidence computation is doing using the *error* of an equivalence partition (Definition 4). We then obtain that $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{rep})$ is equal to

$$1 - \frac{\sum_{\mathsf{eq} \in \Pi(\mathsf{X}, t_p[\mathsf{X}])} \mathsf{error}(\mathsf{eq}, \Pi(\mathsf{X} \cup \{A\}))}{\|\Pi(\mathsf{X}, t_p[\mathsf{X}])\|}$$

We note that $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{rep})$ is computed when element $(\mathsf{X} \cup \{\mathsf{A}\}, t_p)$ is considered. By contrast to CTane and our algorithms from Chapter 4, element $(\mathsf{X}, t_p[\mathsf{X}])$, needed to compute $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{rep})$, 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 16) 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 $|\Pi(\mathsf{A}, t_p[\mathsf{A}])^{\mathsf{eq}}| > 1$. 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 $\Pi(\mathsf{A}, t_p[\mathsf{A}])^{\mathsf{eq}}$. Such a check can be easily integrated during the confidence computation of φ.

Conditions (1) and (2) in Definition 16 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 $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty})$ 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 $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty})$ and $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{rep})$, 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}. We show the pseudocode of the algorithm in Algorithm 8. The PULLBACK procedure goes as follows: For an equivalence class $\mathsf{eq} \in \Pi(\mathsf{Y}, s_p)$, let $\Delta\mathsf{eq}$ be the set of tids in eq that underwent changes in attributes in Y, consistent with the pattern tuple s_p. That is, $\Delta\mathsf{eq}$ contains all tids in eq such that there is a modification $\mathfrak{m} = (\mathsf{tid}, \mathsf{B}, v_d, v_c)$ in $\mathfrak{M}$ such that $\mathsf{B} \in \mathsf{Y}$, $v_c \asymp s_p[\mathsf{B}]$ and $\mathsf{tid} \in \mathsf{eq}$. We initialize $\Pi_d(\mathsf{Y}, s_p)$ by adding an equivalence class $\mathsf{eq}_d = \mathsf{eq} \setminus \Delta\mathsf{eq}$, for each $\mathsf{eq} \in \Pi(\mathsf{Y}, s_p)$. Since each tid in eq_d corresponds to a tuple in D_{dirty} that has not been changed in positions relevant to (Y, s_p), we have that for

any two tids, say tid and tid′, in eq_d, $s[\mathsf{Y}] = s'[\mathsf{Y}] \asymp s_p$ for $s = D_{dirty}[\mathsf{tid}]$ and $s' = D_{dirty}[\mathsf{tid}']$. In other words, eq_d is a proper equivalence class for D_{dirty}.

Those tuples $t \in D_{dirty}$ corresponding to a $\mathsf{tid} \in \Delta\mathsf{eq}$, for some $\mathsf{eq} \in \Pi(\mathsf{Y}, s_p)$, still need to be assigned to their correct equivalence class in $\Pi_d(\mathsf{Y}, s_p)$, or even to a new equivalence class, in case no correct class can be found. More precisely, for each $\mathsf{eq} \in \Pi(\mathsf{Y}, s_p)$ and each $\mathsf{tid} \in \Delta\mathsf{eq}$, we look up $t[\mathsf{Y}]$ for $t = D_{dirty}[\mathsf{tid}]$, and add tid to the equivalence class eq_d corresponding to tuples $s \in D_{dirty}$ with $s[\mathsf{Y}] = t[\mathsf{Y}]$, if such eq_d exists. Otherwise, we create a new equivalence class for tuples in D_{dirty} agreeing with t on its Y-attributes.

Algorithm 8 Pulling back an equivalence class $\Pi(\mathsf{Y}, s_p)$ in D_{rep} to its counterpart $\Pi_d(\mathsf{Y}, s_p)$ in D_{dirty}

```
1:  procedure PULLBACK(Π(Y, s_p), D_dirty, 𝔐)
2:      Π_d(Y, s_p) ← ∅
3:      for all eq ∈ Π(Y, s_p) do
4:          Compute Δeq, the set of modified tids in eq
5:          eq_d ← eq \ Δeq
6:          Π_d(Y, s_p) ← Π_d(Y, s_p) ∪ {eq_d}
7:      Δ ← {Δeq | eq ∈ Π(Y, s_p)}
8:      for all tid ∈ Δ do
9:          eq_d ← FINDEQ(tid, Π_d(Y, s_p), D_dirty)
10:         if eq_d ≠ ∅ then
11:             eq_d ← eq_d ∪ {tid}
12:         else
13:             Create a new eq_d, initially empty
14:             eq_d ← {tid}
15:             Π(Y, s_p) ← Π_d(Y, s_p) ∪ {eq_d}
16:     return Π_d(Y, s_p)
```

5.4 Approximating The Score

As already observed in Section 5.2.3, the computation of $\mathsf{score}(\varphi, \mathfrak{M})$, as defined in Definition 17, 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 UC-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 $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leqslant \mathsf{score}(\varphi, \mathfrak{M})$ and hence, $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leqslant \mathsf{UB}_0(\mathsf{X}, t_p)$ (or $\mathsf{UB}_\lambda(\mathsf{X}, t_p)$). This implies that XPLODE can also be used for computing the global explanation with maximal UC-score.

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

5.4.1 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. 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 19 ($\mathsf{UC\text{-}score}(\cdot)$, constant CFD)**.** Let φ be a constant CFD and $\mathfrak{M}$ a set of modifications. We define $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M})$ as

$$\max\{|M| \mid M \subseteq \mathfrak{M} \text{ is valid and each } \mathfrak{m} \in M \text{ is locally explained by } \varphi\}. \qquad \square$$

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 11. Consider modification $\mathfrak{m}_3 = (8, \mathsf{CC}, 44, 01)$ from the running example. A variable CFD that locally explains this modification is $\varphi = (\mathsf{CC} \rightarrow \mathsf{PN}, (\_, \_))$. 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 $\varphi = (\mathsf{X} \rightarrow \mathsf{A}, (t_p, \_))$ 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 20 ($\mathsf{UC\text{-}score}(\cdot)$, union of constant 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 $\mathsf{UC\text{-}score}(\Sigma_\varphi, \mathfrak{M})$ as

$$\max\{|M| \mid M \subseteq \mathfrak{M} \text{ is } \Sigma\text{-valid and each } \mathfrak{m} \in M \text{ is locally explained by } \varphi\}. \qquad \square$$

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 $\varphi = (\mathsf{X} \rightarrow \mathsf{A}, (t_p, \_))$. We convert φ into a union Σ_φ of constant CFDs as follows: for each equivalence class

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

Algorithm 9 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
```

$\mathsf{eq} \in \Pi_d(\mathsf{X}, t_p)$, 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 5.3.4) is computed by "removing tuples that do belong to classes in $\Pi(\mathsf{A}, a)^{\mathsf{eq}}$, 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 21 (UC-score$(\cdot)$ of a CFD)**.** Let φ be a CFD, $\Sigma = \{\varphi_1, \cdots, \varphi_m\}$ a union of constant CFDs, and $\mathfrak{M}$ a set of modifications. We define

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

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

5.4.2 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

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

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 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 4. For any Σ_φ-valid set of modifications $M \subseteq \mathfrak{M}$, M is locally explained by Σ_φ if and only if each $\mathfrak{m} \in M$ 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. □

The proof of this proposition is presented in Appendix B.1. 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 5.2.3. An important property is that:

Proposition 5. For every global explanation φ and set $\mathfrak{M}$ of modifications, $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leqslant \mathsf{score}(\varphi, \mathfrak{M})$.

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

The full proof of this proposition is given in Appendix B.2. Consequently, $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M}) \leqslant \mathsf{score}(\varphi, \mathfrak{M}) \leqslant \mathsf{UB}_0(\mathsf{X}, t_p)$ (and $\mathsf{UB}_\lambda(\mathsf{X}, t_p)$) when φ is of the form $(\mathsf{X} \setminus \mathsf{A} \to \mathsf{A}, t_p)$; hence XPLODE finds the global explanation of highest UC-score.

5.4.3 Computation of UC-score

We conclude by explaining how $\mathsf{UC\text{-}score}(\varphi, \mathfrak{M})$ can be efficiently computed. Proposition 4 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 6. Let Σ_φ be the set of constant CFDs corresponding to CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$. Let $\mathfrak{m} = (\mathsf{tid}, \mathsf{B}, v_d, v_c) \in M$, with $M \subseteq \mathfrak{M}$ a Σ-valid set of modifications, $s = D_{dirty}[\mathsf{tid}]$ and $t = (D_{dirty} \oplus \mathfrak{m})[\mathsf{tid}]$. Then Σ_φ locally explains $\mathfrak{m}$ 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).* □

The proof of this proposition is presented in Appendix B.3. Of course, for constant CFDs φ, $\Sigma_\varphi = \{\varphi\}$ and hence only cases (1) and (3) in the Proposition apply. Although Definition 16 also requires checking whether

$\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty}) < \mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})$, as part of the proof of Proposition 6 we show that this is implied by the conditions in its statement.

The pseudo-code for computing UC-score, shown in Algorithm 9, is based on Propositions 4 and 6. 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 6 it suffices to only consider $\mathfrak{m} \in \mathfrak{M}$ 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 $\mathfrak{m} = (\mathsf{tid}, \mathsf{B}, v_d, v_c)$ 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 6); 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 6). 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 6). As soon as a modification is explained for a given tid t, it is counted, and the algorithm proceeds to the next tid.

5.5 Experiments

We experimentally validate our repair explanation method. 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. The code, datasets and CFDs used, are available for research purposes [92].

5.5.1 Experimental Setup

Datasets. Statistics of the data are shown in Table 5.2. To ensure that different kinds of CFD violations can occur, and we test our method on a variety of repairs, we duplicate every tuple in each of the datasets. On the Adult dataset we only use constants CFDs, since mining general CFDs on this dataset was too time-consuming using the traditional CTane algorithm. This is due to the higher number of attributes in the Adult dataset with, on average, around 10 values in their domain. Since most of these values have a high support, and their combinations as well, this leads to many frequent constant patterns.

Error Generation. We make use of the BART tool [8] 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 5.4. To get the required quality rules, we used our implementation of CTane [46] to discover CFDs on the datasets, using the minimum support percentages shown

Table 5.2: 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%

Table 5.3: Position of target CFD among all approximate CFDs according to various ranking criteria.

Dataset	**CFD**	**Length** (↑)	**Conf** (↑)	**Conf** (↓)	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

in Table 5.2. As a note aside, the support threshold δ is then simply computed as $\frac{1}{100}$(**#Tuples**$\times$**%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. In contrast to the previous chapter, no limitations were put on the antecedent size of the CFDs. 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. We denote the used CFDs 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 that we want to discover by repairing the corresponding dirty dataset. Obviously, CFD *i* is different for each dataset.

Falcon. We compare with FALCON [58], 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.

5.5.2 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. For this experiment, we perform regular CFD discovery on the dirty data, and rank the discovered approximate CFDs by confidence and rule length. We compare these baseline approaches to a ranking of global explanations using our scoring function $\mathsf{UC\text{-}score}(\cdot)$. Since our approach requires modifications, we compute this ranking on two partially cleaned versions of the data, with 2 and 5 modifications, respectively.

In Table 5.3, 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, i.e., the "best explanation"). The results clearly show that, when using confidence or rule size, the target CFD is typically quite deep in the ranking. Of course, these uninformed baselines have no way of knowing which data is dirty, and hence whether a CFD is useful for repairing. It is clearly 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 and quickly brings the target to the front, ratifying our approach. By manually repairing a handful of tuples, many CFDs are automatically invalidated, and hence a large amount of user effort is saved.

5.5.3 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}, as opposed to partial repairs. 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 5.2. 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. We report the number of candidate global explanations on the various datasets, for different numbers of modifications, in Appendix B.4. Secondly, we want

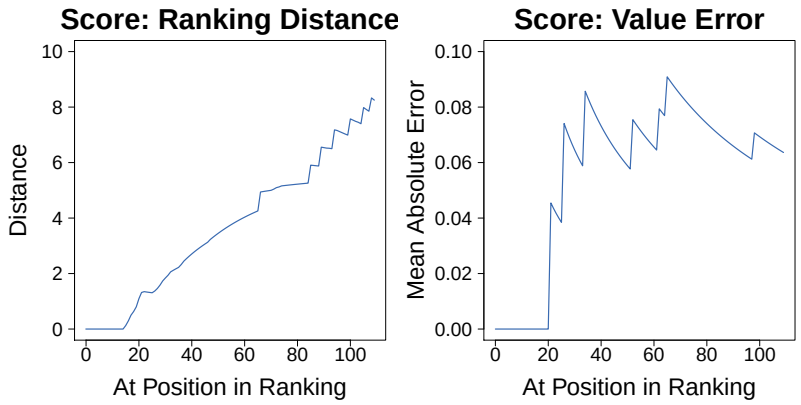


Figure 5.2: Comparison of UC-score(·) and score(·) on the Abalone dataset, using 20 modifications.

to validate that the target CFD is indeed the highest scoring (for UC-score(·)) global explanation when considering full repairs. Indeed, on all datasets the target CFD was discovered. It shows that the UC-score(·) is a good measure for repair explanations.

5.5.4 Scoring Function

We now further evaluate whether the scoring function UC-score(·) is indeed a suitable surrogate for score(·), the exact scoring function. Since score(·) requires an expensive computation, which we implemented using a brute-force approach, we perform the experiment only on the smallest dataset, Abalone. We mined all global explanations on the data with a 1% error percentage, using a partial repair containing 20 modifications. Both UC-score(·) and score(·) report values ranging from 1 to 20 for the global explanations. Again, since we need all global explanations, we make use of the naive post-processing method. We rank all global explanations, using both scoring functions. At each point in the ranked list, we compute the Mean Spearman's Footrule Distance and the Mean Absolute Error of the scores, for all explanations up to this point in the ranking. The results are shown in Figure 5.2. 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. The sea-saw pattern is a result of averaging: when a difference is encountered, the error rises, and then smoothens out as additional positions, without error, are considered. Hence, UC-score(·) offers a good approximation for score(·), and is unlikely to change which global explanation is returned by XPLODE, compared to using score(·). We therefore use UC-score(·) in all remaining experiments.

Table 5.4: Number and Percentage of Modifications required to retrieve the target CFD, for 3 different CFDs.

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

5.5.5 Explaining Partial Repairs

Our previous experiments showed that the target CFD, the one used to dirty the data, can be recovered from a full repair of the data, using our scoring function UC-score$(\cdot)$. We now want to answer the crucial question, "how many modifications are needed for a partial repair to recover the target CFD?". To answer this question, we created partial repairs using an increasing number of modifications, and report the number and percentage of modifications needed until the target CFD was returned by XPLODE. 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 5.4. 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 found by manually cleaning 1% of the violations, then 99% of the violations may be cleaned automatically. Moreover, it seems that the number of attributes has a higher impact on the number of required modifications than the number of tuples.

5.5.6 Comparison with Falcon

We next compare XPLODE and FALCON [58], for the case where the target CFD is a constant CFD[5]. 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 when considering arbitrary modifications (not in the consequent) or variable CFDs. Indeed, this would require FALCON to obtain a candidate CFD for every changed value in the antecedent, or for every value combination 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 modifications involving attributes in the antecedent of CFDs are present.

5.5.7 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". For this experiment, we consider again a full repair, as in our first experiment. Additionally, we add a number of random modifications throughout the data, that have no connection to the target CFD. In Figure 5.3, 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.

[5]Due to IP restrictions, we were unable to obtain the code and perform an in-depth comparison. One of the authors of FALCON kindly performed an experiment instead.

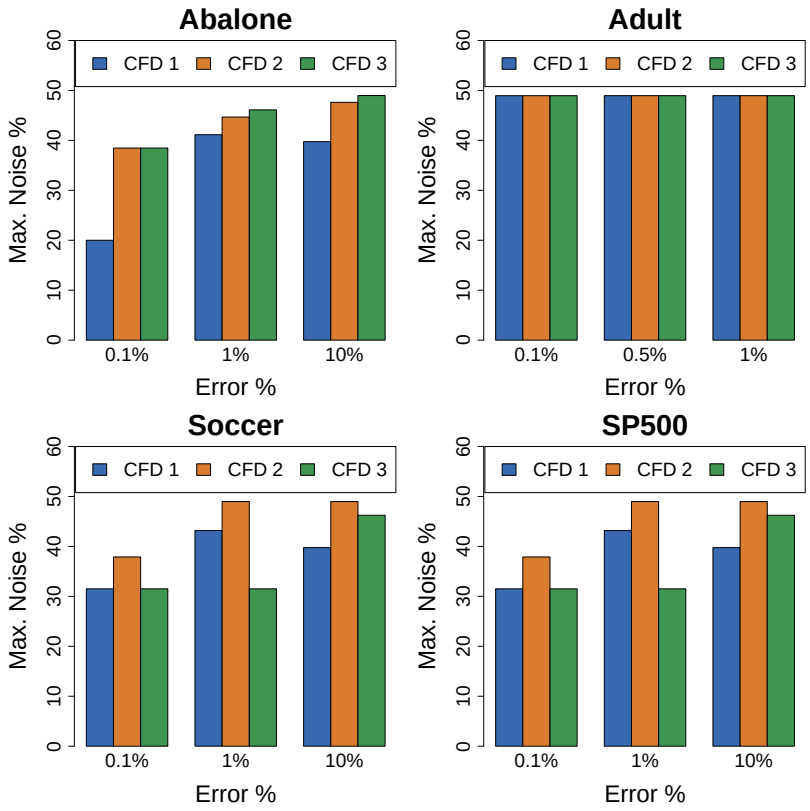


Figure 5.3: Noise-robustness of XPLODE.

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 will not be able to connect the random modifications to each other.

5.5.8 Runtime Performance

We now turn our attention towards the performance of XPLODE. The runtime was evaluated 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 5.4 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 de-

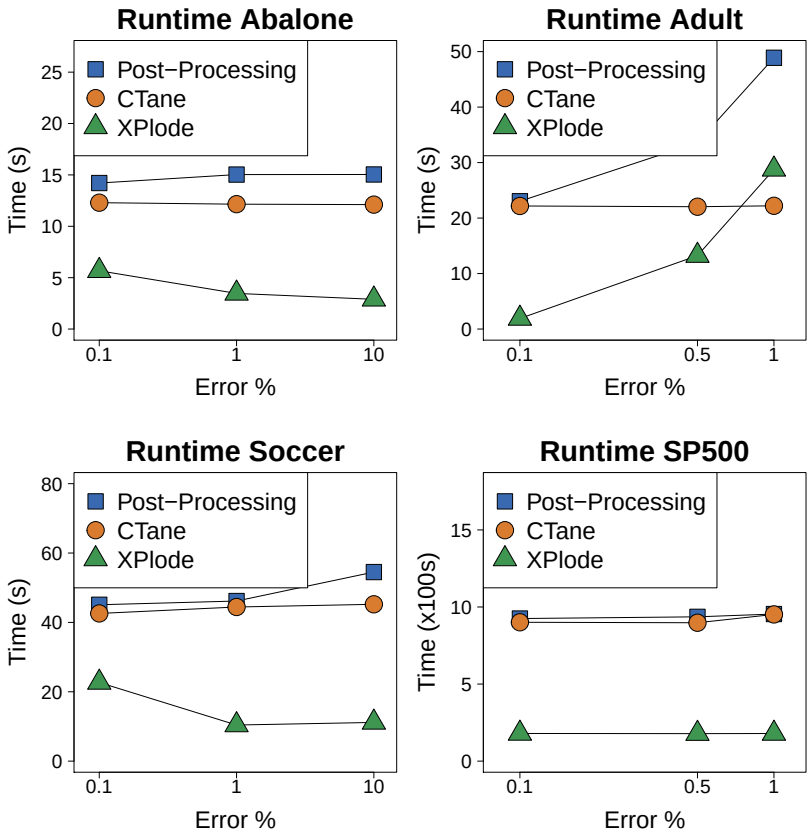


Figure 5.4: Runtime performance of XPLODE, compared to post-processing and CTane.

teriorates 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.

5.5.9 Influence of Optimizations

Finally, we evaluate the impact of three optimizations discussed throughout Section 5.3. We compare the runtimes with each of these optimizations turned off in turn, to the runtime of XPLODE with all optimizations turned on. The runtimes were again obtained using a full repair, as in the runtime experiment. The results are shown in Figure 5.5. The worst runtime is obtained when disabling "All scores", which is the computation of scores for CFDs that are not global explanations (discussed under Remarks at the end of Section 5.3.1). Computing these scores leads to better informed tiebreak-

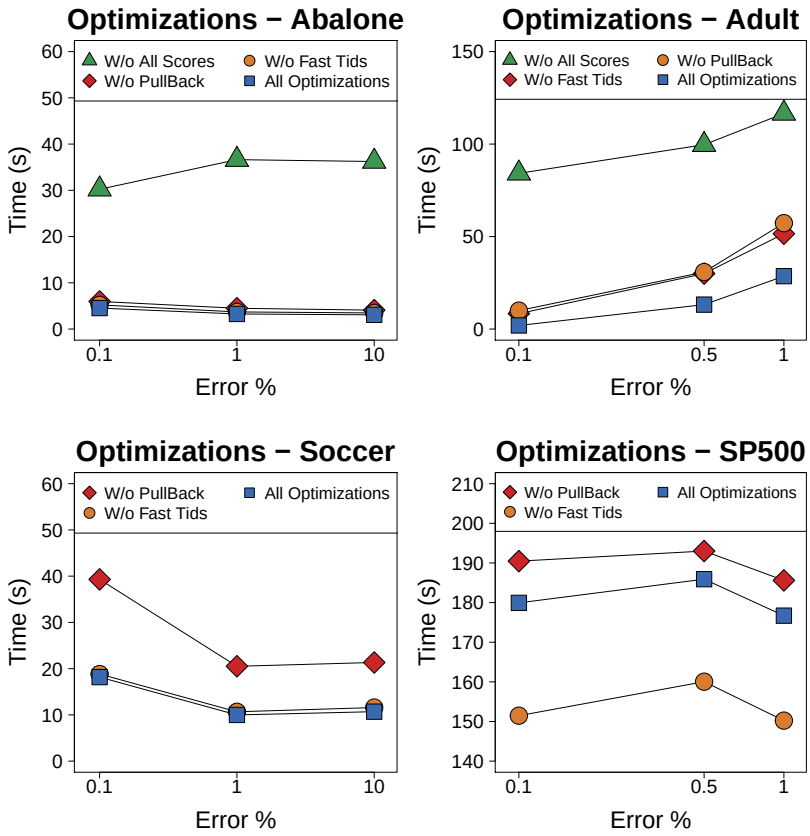


Figure 5.5: Impact of optimizations on runtime.

ing among elements with equal upper bounds, at the cost of having to compute the score. Clearly, computing all scores leads to a major improvement in runtime. For this reason, we do not test without this optimization on the larger datasets, Soccer and SP500. The other two optimizations concern the method of obtaining equivalence partitions by XPLODE. Recall that, when computing the confidence of a CFD, the equivalence partition of a subset may not be available. We evaluate as "Fast Tids" the computation of these equivalence partitions on-the-fly from the partitions of singletons, as opposed to computing them from the data. This optimization has a positive impact on runtime, on all datasets except SP500. Finally, we evaluate the use of the PULLBACK method, to convert an equivalence partition on D_{rep} to its counterpart on D_{dirty}. PULLBACK offers the least benefit of our optimizations, but still leads to a clear and consistent improvement. The percentage of errors does not seem to change the relative influence of the optimizations.

5.6 Related Work

In this chapter, we have relied on users to make some initial repairs, after which existing data repairing algorithms can be used for cleaning the data. We have overviewed such repair methods in Chapter 3. Our approach is complementary to any of these repairing algorithms. In this section, we discuss the work that is specifically related to our discovery of explanations.

5.6.1 Traditional Constraint Discovery

Our algorithm XPLODE is built upon an underlying constraint discovery process. Traditional (C)FD discovery algorithms, such as the ones presented in the previous Chapter [93], or those presented in Chiang and Miller [29], Fan et al. [46], Huhtala et al. [60], Papenbrock et al. [85], and Mandros et al. [75], aim to find *all* constraints that approximately hold on the data. In contrast, we aim to find only those CFDs that *explain a partial repair*, in order to restrict the output of the discovery algorithm to *useful* constraints for repairing. Traditional methods do not consider our notions of explanation and scoring function during the discovery process. To find explanations, these algorithms therefore have to be combined with a post-processing step, which is expensive both in time and user interaction. In our algorithm XPLODE, post-processing is avoided by carefully integrating the notion of explanations in the discovery process.

Our resulting on-demand algorithm is similar in spirit to the method presented in Golab et al. [55], where constants patterns for a *fixed* FD are found that best describe the data. The efficiency of their algorithm is based on the anti-monotonicity of the support and confidence measures. In contrast, our search strategy allows for general *loose anti-monotonic* functions [19], which include a much larger number of measures, such as our scoring function. In addition, we explore the *entire* space of CFDs instead of requiring the embedded FD to be fixed. We leverage equivalence partitions, introduced in Cosmadakis et al. [36], and commonly used in constraint discovery [60, 46]. None of the traditional constraint discovery methods consider user interaction, which is *essential* for our method.

Whereas the on-demand algorithm is currently based on the Tane algorithm, it could readily be adapted to make use of recent advances in CFD discovery such as Kruse and Naumann [69] and Rammelaere and Geerts [93]. Moreover, our method naturally lends itself to the hybrid approach taken in Papenbrock and Naumann [84].

In Berti-Équille et al. [15], the set of constraints is further restricted by discovering *genuine* FDs, based on measuring the probability that the constraint will be satisfied after some repair is applied. However, it is not guaranteed that the FDs with the highest genuineness, are also useful for repairing.

Related to our explanation, in Chalamalla et al. [26], a *description of errors* by means of a small set of conjunctive queries is discovered, without considering user interaction or repairing. Moreover, our repair explanations could also be of use to update constraints over evolving data, in line with the meth-

ods presented in Chiang and Miller [30], Beskales et al. [16], and Mazuran et al. [78].

5.6.2 User-guided Constraint Discovery

Other discovery methods that leverage user interaction are presented in He et al. [58] and Thirumuruganathan et al. [102].

The FALCON system [58], described in more detail in Section 5.5.1, 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 of the system is on limiting calls to this oracle. By contrast, we *only* use information stemming from *multiple* modifications, and as such our method has no need for a "black box" oracle. Moreover, 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 [102] 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.

5.6.3 User Interaction in Data Cleaning

Other forms of user interaction in data cleaning are considered in works such as Wang et al. [105], Volkovs et al. [103], and Yakout et al. [110]. In these works, the set of (valid) constraints (FDs, CFDs, or other) is assumed to be available already. User involvement is then used to determine the right repairs relative to these constraints. In contrast, our work leverages user interaction in an earlier phase, to *find valid constraints for repairing*. Once these constraints have been found, user interaction can again be integrated into the repair process.

5.7 Conclusion

In this chapter, we have focused on the problem of involving a non-expert user into a constraint discovery process. More concretely, we considered the problem of finding a CFD that best explains an observed partial repair. We have presented an on-demand algorithm, XPLODE, that 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, when compared to a post-processing approach to the considered problem.

With extensive experiments, we show that our approach is suited for discovering valid CFDs that are useful for repairing. Our method shows high precision in discovering the correct explanation, using only a small amount of modifications. Moreover, we have verified that XPLODE is considerably faster than the naive, post-processing approach. Additional experiments show that our method is very nobust to noise in the observed modifications, and that our optimizations, specific to the task of discovering explanations, further decrease the runtime of XPLODE.

Future work includes the investigation of alternative scoring functions, and extending XPLODE towards discovering multiple explanations. The general concept of inferring constraints from a repair can be investigated in the context of various other constraint formalisms. Finally, it would be interesting to see how our method can be integrated with other forms of user interaction, for example by suggesting modifications to a user, and incorporating their responses into our algorithm.

CHAPTER 6

Forbidden Itemsets

6.1 Introduction

So far in this dissertation, we have considered the discovery of *constraints* for the purpose of cleaning data, with or without the involvement of a human user. We now consider, instead, the direct discovery of *inconsistencies*, i.e., the violations of some constraint. Moreover, we now focus on *automatic* cleaning, such that error detection and subsequent repairing can be performed without any necessary input from the human. In this chapter, we introduce a dynamic notion of data quality for use in such an automatic context. In this chapter we first focus on the discovery of inconsistencies, before presenting a repair method based on the dynamic notion of data quality in the next chapter.

Consider a scenario where a user runs a constraint discovery algorithm on a dataset, with the intent on cleaning said data. Many algorithms are available for discovering such constraints. However, since the underlying data is dirty, this raises concerns about the reliability of the discovered constraints. Assume for the moment that the constraints are reliable and are used to repair (clean) a dirty database. For the sake of the argument, what if one re-runs the constraint discovery algorithm on the repair and finds other constraints? Does this imply that the repair is not clean after all, or that the discovery algorithm may in fact find unreliable constraints? It is a typical chicken or the egg dilemma. The problem is that constraints are considered to be *static*: once found, they are treated as a gold standard. To remedy this, we propose a *dynamic* notion of data quality. The idea is simple:

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

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

The main contribution of chapters 5 and 6 is to illustrate this dynamic view on data quality for a particular class of constraints. In particular, we consider errors that can be caught by so-called *edits*, which is "the" constraint formalism used by census bureaus worldwide [48, 59] and can be seen as a simple class of denial constraints [31, 32]. Intuitively, an edit specifies *forbidden value combinations*. For example, an age attribute cannot take a value higher than 130, a city can only have certain zip codes, and people of young age cannot have a driver's license. These edits are typically designed by domain experts and are generally accepted to be a good constraint formalism for detecting errors that occur in single records. In contrast, we aim to automatically discover such edits in an unsupervised way by using pattern mining techniques. To make the link to pattern mining more explicit and to differentiate from edits, we call our patterns *forbidden itemsets*. Although our technique is designed such that human supervision is not compulsory, we show that optional user interaction can be readily integrated to improve the reliability of the methods.

Pattern mining techniques are typically used to uncover positive associations between items, measured by different interestingness measures such as frequency, confidence, lift, and many others. Based on experience, *discovered patterns often reveal errors in the data* in addition to interesting associations. For example, an association rule which holds for 99% of the data could be interesting in itself, but might also represent a well-known dependency in the data which should hold for 100%. The fact that the association does not hold for 1% of the data is then more interesting. This 1% of the data often points to unlikely co-occurrences of values in the data, which forbidden itemsets aim to capture. In order to reliably detect unlikely co-occurrences, it is clear that a large body of clean data is needed. We therefore focus on low error rate data, such as census data or data that underwent some curation [24].

The biggest challenge given our dynamic notion of data quality, however, is to provide suggestions for how to repair the discovered errors. We discuss this in the next chapter.

Figure 6.1 gives a schematic overview of the proposed cleaning mechanism in its entirety. We capture unlikely co-occurrences by means of forbidden itemsets. Our algorithm FBIMINER employs pruning strategies to optimize the discovery of forbidden itemsets. Linking back to the beginning of the introduction, we will regard data to be dirty if FBIMINER finds forbidden itemsets in the data. Users may optionally filter out forbidden itemsets by declaring them as valid. Furthermore, we also devise a *repairing algorithm* that repairs a dirty dataset and ensures that no forbidden itemsets ex-

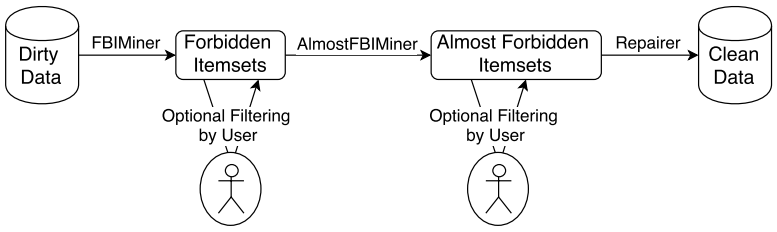


Figure 6.1: Schematic overview of the proposed FBI cleaning mechanism.

ist in the repair, hence it is indeed clean. To achieve this, we consider *almost forbidden itemsets* and present an algorithm A-FBIMINER for mining them. Again, users can optionally filter out such itemsets. Both the discovery of almost forbidden itemsets, and the repair strategy, are discussed in the next chapter.

6.1.1 Summary of Contributions

1. We formalize the dynamic notion of data quality in full generality, stating that a dataset is considered clean if a constraint discovery algorithm does not detect any violations in the data. (Section 6.2)

2. We introduce the concept of *forbidden itemsets*, which represent illegal value combinations found in a dataset, using a general likeliness function. We then instantiate this general framework for the specific case of low-lift forbidden itemsets, and provide examples for the effectiveness of this new formalism. (Sections 6.3)

3. We provide an efficient itemset mining algorithm for discovering forbidden itemsets. The efficiency is based on pruning properties, which we derive from the lift measure that is used. (Section 6.4)

4. We experimentally evaluate our method on various real-world datasets, showing that the discovered forbidden itemsets can detect errors with high precision, and that the discovery algorithm is highly efficient. (Section 6.7)

6.2 Problem Statement

We first phrase our problem in full generality (following Mannila and Toivonen [76]) before making things more specific in the next section. Consider a transaction dataset $\mathcal{D}$ and some *constraint language* Γ for expressing properties that indicate *dirtiness* in the data, e.g., Γ could consist of conditional functional dependencies, association rules, edits, or denial constraints, among others. Furthermore, let q be a *selection predicate* (evaluating to true or false) that assesses the relevance of constraints $\varphi \in \Gamma$ in $\mathcal{D}$. For example, when φ is a conditional functional dependency, $q(\mathcal{D}, \varphi)$ may return true if φ is violated in $\mathcal{D}$.

We denote by $\mathsf{dirty}(\mathcal{D},\Gamma,q)$ the set of all *dirty constraints*, i.e., all $\varphi \in \Gamma$ for which $q(\mathcal{D},\varphi)$ evaluates to true. For example, $\mathsf{dirty}(\mathcal{D},\Gamma,q)$ may consist of all violated conditional functional dependencies, all edits that apply to an object, or all low confidence association rules.

Definition 22. A dataset $\mathcal{D}$ is said to be *clean* relative to language Γ and predicate q if $\mathsf{dirty}(\mathcal{D},\Gamma,q)$ is empty; $\mathcal{D}$ is called *dirty* otherwise. □

With this definition we take a *completely new view* on data quality. Indeed, existing work in this area typically *fixes* the constraints up front, regardless of the data [44]. For example, edits are often designed by experts and then compared with the data. In our definition, we only specify the *class* of constraints, e.g., edits. Which edits are used for declaring the data clean or dirty *depends entirely on the underlying data*. We thus introduce a *dynamic* rather than a static notion of dirtiness/cleanliness of data: when the data changes, so do the edits under consideration. With this notion at hand, we are next interested in repairs of the data. Intuitively, a repair of a dirty dataset is a modified dataset that is clean.

Definition 23. Given datasets $\mathcal{D}$ and $\mathcal{D}'$, language Γ, predicate q and similarity function sim between datasets, we say that $\mathcal{D}'$ is an (Γ,q)*-repair* if (i) $\mathcal{D}'$ has the same set of object identifiers as $\mathcal{D}$; and (ii) $\mathsf{dirty}(\mathcal{D}',\Gamma,q)$ is empty. An (Γ,q)-repair $\mathcal{D}'$ is *optimal* if $\mathsf{sim}(\mathcal{D},\mathcal{D}')$ is maximal amongst all (Γ,q)-repairs of $\mathcal{D}$. □

Here, we assume that a similarity measure between objects is given, and denote by $\mathsf{sim}(o,o')$ the similarity between objects o and o'. The similarity between two datasets $\mathcal{D}$ and $\mathcal{D}'$ is denoted by $\mathsf{sim}(\mathcal{D},\mathcal{D}')$ and is defined by

$$\frac{1}{|\mathcal{D}|} \sum_{o=\langle \mathsf{tid},I\rangle \in \mathcal{D}, o'=\langle \mathsf{tid},J\rangle \in \mathcal{D}',} \mathsf{sim}(o,o').$$

We observe that the sum ranges over pairs of objects in $\mathcal{D}$ and $\mathcal{D}'$ with the same object identifier. Furthermore, the choice of similarity function is not crucial for our algorithms: Any similarity measure can be used in our setting. Of course, the quality of repairs will depend on the chosen measure.

6.3 Forbidden Itemsets

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

Definition 24. Let $\mathcal{D}$ be a dataset, L a likeliness function, I an itemset and τ a maximum likeliness threshold. Then, I is an (L,τ)-*forbidden itemset* whenever $L(I,\mathcal{D}) \leqslant \tau$. □

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

To make things even more specific, we fix the likeliness function L to be the *lift measure* such that $\mathsf{dirty}(\mathcal{D},\Gamma,q)$ corresponds to itemsets that do occur in the data, despite being very improbable with respect to the rest of the data. We will illustrate the effectiveness of using lift as a likeliness function shortly, and show examples of discovered forbidden itemsets in real datasets, in Section 6.3.2. In the following, when L is the lift measure we refer to (L,τ)-forbidden itemsets simply as τ-*forbidden itemsets*. We next make this more formal.

6.3.1 Lift Measure for Itemsets

Intuitively, the lift measure indicates how probable the co-occurrence of a set of items is given their separate frequencies. Lift is generally used as an interestingness measure in association rule mining, where rules with a high lift between antecedent and consequent are considered the most interesting [23]. Recall from Chapter 2 that the lift of an association rule $I \to J$ is given by

$$\mathsf{lift}_{\mathsf{AR}}(I \to J,\mathcal{D}) := \frac{\mathsf{freq}(I \cup J,\mathcal{D})}{\mathsf{freq}(I,\mathcal{D}) \times \mathsf{freq}(J,\mathcal{D})}.$$

Following Webb and Vreeken [108], we then define the *lift of an itemset* as

$$\mathsf{lift}(I,\mathcal{D}) = \max_{\emptyset \subset J \subset I} \mathsf{lift}_{\mathsf{AR}}\big((I \setminus J) \to J,\mathcal{D}\big).$$

It will be convenient to write this definition more explicitly in terms of supports of itemsets.

Definition 25. Let $\mathcal{D}$ be a dataset and I be an itemset. The *lift* of I, denoted by $\mathsf{lift}(I,\mathcal{D})$, is defined as

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

In other words, we adopt a lift measure based on "pairwise" independence, in which $\mathsf{freq}(I,\mathcal{D})$ is compared against the product $\mathsf{freq}(J,\mathcal{D}) \times \mathsf{freq}(I \setminus J,\mathcal{D})$ for any non-empty $J \subset I$, and the *maximal* discrepancy ratio is taken as the *lift* of I in $\mathcal{D}$. As mentioned earlier, we take the lift measure as our likeliness function. In accordance with Def. 24 we then have that:

Definition 26. A τ-*forbidden itemset* is an itemset I for which $\mathsf{lift}(I,\mathcal{D}) \leqslant \tau$ holds. □

We observe that when using lift, τ will typically be small, and we assume that $\tau < 1$. Setting $\tau \geqslant 1$ would imply positive (or neutral) correlations among the items, which makes little sense for detecting unlikely value combinations. In the next chapter of this dissertation (Section 7.7) we further restrict τ such that $\tau < 3/4$. This is needed to guarantee correctness of our repair algorithm. This restriction is harmless from a practical point of view, however, since sensible forbidden itemsets have a lift measure much smaller than $3/4$.

With these definitions at hand, we can now formalize the problem of discovering low-lift τ-forbidden itemsets:

PROBLEM: Discovering τ-forbidden itemsets

INPUT: Dataset $\mathcal{D}$, lift threshold τ

OUTPUT: All itemsets I in $\mathcal{D}$ with $\mathsf{lift}(I,\mathcal{D}) \leqslant \tau$.

6.3.2 Effectiveness of Forbidden Itemsets

We here provide some initial motivation for considering invalid or unlikely value combinations (formalized by forbidden itemsets) as an error detection formalism. To illustrate that τ-forbidden itemsets are an effective formalism for capturing unlikely value combinations, we start with an example.

Example 12. Consider the example forbidden itemsets found in some UCI datasets [39], as shown in Table 6.1. In the Adult dataset, the co-occurrence of Female and Husband, as well as Male and Wife, are clearly erroneous. Other examples involve a married person under the age of 18 and people who are married, yet living in with an unrelated household. In the Zoo dataset, the first forbidden itemset shows that the animal *clam* in the dataset is neither aquatic nor breathing. To our knowledge, clams are in fact aquatic, so these values are indeed in error. The other forbidden itemsets detect animals that are in some way an exception in nature. For example, the *platypus* is famous for being one of few existing mammal species that lays eggs, the other species being anteaters. Similar exceptional combinations are encountered in the other UCI datasets, such as the Mushroom dataset, although the forbidden itemsets are more difficult to interpret for this dataset. In the Students dataset, we detect instances where alcohol consumption is higher on weekdays than during the weekend, which seems counter-intuitive. In the GermanCredit dataset, the forbidden itemset (JOB=UNSKILLED, EMPLOYEDSINCE=UNEMPLOYED) is erroneous since the Job attribute has a specific value for unemployed people. While not all of these examples represent actual *errors*, they show that forbidden itemsets are capable of detecting peculiar objects that require extra attention. Of course, it makes little sense to repair objects such as the platypus. Typically, user validation of the discovered errors will be preferable over fully automatic repairs. This is addressed in Section 6.5. $\diamondsuit$

Table 6.1: Example forbidden itemsets found in UCI datasets.

Forbidden Itemsets	**Dataset**	τ
Sex=Female, Relation=Husband Sex=Male, Relation=Wife Age=<18, Marital-status=Married Age=<18, Relationship=Husband Relation=Not-in-family, Marital=Married	Adult	0.01
aquatic=0, breathes=0 (clam) type=mammal, eggs=1 (platypus) milk=1, eggs=1 (platypus) type=mammal, toothed=0 (platypus) milk=1, toothed=0 (platypus) eggs=0, toothed=0 (scorpion) tail=1, backbone=0 (scorpion)	Zoo	0.1
bruises=t, habitat=l population=y, cap-shape=k cap-surface=s, odor=n, habitat=d cap-surface=s, gill-size=b, habitat=d edible=e, habitat=d, cap-shape=k	Mushroom	0.025
Job=Unskilled,EmployedSince=Unemployed Property=Real Estate,Housing=Free Property=Life Insurance,Housing=Free Property=Car or Other, Housing=Free	GermanCredit	0.1
WeekAlc=1, DayAlc=2 WeekAlc=1, DayAlc=3 FatherEdu=High, MotherEdu=Low FatherEdu=Low, MotherJob=Teacher	Students	0.075

In fact, forbidden itemsets capture a considerable amount of errors commonly found by other formalisms. Indeed, invalid value combinations have been used for decades to detect errors in census data starting with the seminal work by Fellegi and Holt [48]. Furthermore, although more expressive formalisms such as conditional functional dependencies (CFDs) [45] and denial constraints (DCs) [31, 32] have become popular for error detection and repairing, many constraints used in practice are very simple. As an example, most of the constraints reported in Table 4 in Abedjan et al. [2] can be regarded as constraints that only involve constants. This is clear for "checks" that specify invalid domain values. However, even a functional dependency such as ($zip \rightarrow state$) can be regarded as a (finite) collection of constant rules that associate specific zip codes to state names. The violations of these rules clearly are invalid value combinations. Similarly, almost half of the DCs reported in Chu et al. [32] only involve constants and concern single tuples.

It therefore seems natural to first gain a better understanding of these simple constraints in our dynamic data quality setting. Finally, the discovery of CFDs and DCs [46, 29, 33] in their full generality is very slow due to the high expressiveness of these constraint languages. Indeed, experiments in these papers show that the discovery of general CFDs and DCs may take up to hours on a single machine. This makes general constraints infeasible in settings such as ours, where interactivity or quick response times are needed.

Based on the above, we believe that forbidden itemsets are a suitable method for discovering a set of errors (corresponding with rule violations [2]) with high precision. They provide a good balance between expressiveness, efficiency of discovery, and efficacy in error detection. Furthermore, they are often easily interpretable, allowing users to inspect and filter out false positives, as will be discussed in Section 6.5.

6.4 Discovering forbidden Itemsets

In this section we show how to compute $\mathsf{dirty}(\mathcal{D},\Gamma,q)$ for *low lift* forbidden itemsets. Since low lift itemsets are typically *infrequent*, existing itemset mining algorithms are not optimized for this task. We therefore derive properties of the lift measure that allow for substantial pruning when mining forbidden itemsets (Section 6.4.1). We conclude this section by presenting a version, referred to as FBIMINER, of the well-known Eclat algorithm [116], enhanced with our derived pruning strategies and optimizations specific for the task of mining low lift forbidden itemsets (Section 6.4.2).

6.4.1 Properties of the Lift Measure

Before presenting our algorithm FBIMINER that mines τ-forbidden itemsets, we describe some properties of the lift measure that underlie the pruning strategies of our algorithm. As we will explain shortly, FBIMINER performs a depth-first search for forbidden itemsets. To make this search efficient, pruning strategies should be in place that discard all supersets of a particular itemset, i.e., prune the entire search tree underneath the current node. For this purpose, we derive properties that must hold for all subsets of a τ-forbidden itemset. If such a property does not hold for an itemset encountered in the search tree, then clearly none of its supersets may be τ-forbidden, and this branch of the search space may be pruned.

The development of pruning strategies for the lift measure is quite challenging. Typically, pattern mining algorithms perform pruning by leveraging monotonicity properties of interesting measures. The lift measure, however, is neither monotonic nor anti-monotonic. Nevertheless, since a low lift itemset requires that an itemset occurs *much less often* than its subsets, we can use the relation between the support of a τ-forbidden itemset and the support of its subsets to derive properties for pruning.

In the remainder of this section, we obtain useful bounds on the support of either τ-forbidden itemsets, or their subsets. In general, given a dataset $\mathcal{D}$ and an itemset I occurring in $\mathcal{D}$, it trivially holds that $1 \leqslant \mathsf{supp}(I,\mathcal{D}) \leqslant |\mathcal{D}|$.

However, when discovering itemsets in a specific dataset, we can obtain a tighter upper bound on the support of itemsets, assuming that the support of individual items is known. Indeed, $\mathsf{supp}(I,\mathcal{D}) \leqslant \mathsf{supp}(\{i\},\mathcal{D})$ for any $i \in J$ such that $I \subseteq J$. We define σ_I^{max} to be the highest support of an item $\{i\}$ in I or any of I's supersets J. Note that this can be different from the most frequent item in the data, since an itemset can contain at most one item per attribute, and hence we only consider supersets J such that $J \setminus I$ and I are compatible, i.e., have no overlapping attributes.

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

In other words, σ_I^{max} is the highest support of a single item in I's branch of the search tree. Clearly, it holds that $\mathsf{supp}(I,\mathcal{D}) \leqslant \sigma_J^{max}$, and $\sigma_J^{max} \leqslant \sigma_I^{max}$ for all $I \subset J$.

We first introduce a lower bound on the support of subsets of J, when J is a τ-forbidden itemset:

Proposition 7. For itemsets I and J such that $I \subset J$, if J is a τ-forbidden itemset then $\mathsf{supp}(I,\mathcal{D}) \geqslant \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\sigma_I^{max} \times \tau}$.

Proof. We show this by contradiction. Let J be a τ-forbidden itemset for $\tau < 1$ and assume for the sake of contradiction that there exists a $I \subset J$ with $\mathsf{supp}(I,\mathcal{D}) < \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\sigma_I^{max} \times \tau}$. Then observe the following:

$$\tau \geqslant \mathsf{lift}(J,\mathcal{D}) \geqslant \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\mathsf{supp}(J \setminus I,\mathcal{D}) \times \mathsf{supp}(I,\mathcal{D})}.$$

Using our assumption that $\mathsf{supp}(I,\mathcal{D}) < \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\sigma_I^{max} \times \tau}$, we get

$$\begin{aligned}
\tau &> \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\mathsf{supp}(J \setminus I,\mathcal{D}) \times \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\sigma_I^{max} \times \tau}} \\
&= \frac{\cancel{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})} \times (\sigma_I^{max} \times \tau)}{\mathsf{supp}(J \setminus I,\mathcal{D}) \times \cancel{(|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D}))}} \\
&= \frac{\sigma_I^{max} \times \tau}{\mathsf{supp}(J \setminus I,\mathcal{D})}.
\end{aligned}$$

Since $\mathsf{supp}(J \setminus I,\mathcal{D}) \leqslant \sigma_J^{max} \leqslant \sigma_I^{max}$, we thus have that $\tau > \tau$, which is clearly impossible. Hence, every strict subset I of a τ-forbidden itemset J must satisfy $\mathsf{supp}(I,\mathcal{D}) \geqslant \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D})}{\sigma_I^{max} \times \tau}$. □

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

Proposition 7 can be leveraged to show that a minimum reduction in support between subsets of a τ-forbidden itemset is required. The following proposition allows us to prune the search tree when we encounter an itemset J having a subset whose support is too close to J's support.

Proposition 8. For any three itemsets I, J and K such that $I \subset J \subseteq K$ holds, if K is a τ-forbidden itemset, then $\mathsf{supp}(I,\mathcal{D}) - \mathsf{supp}(J,\mathcal{D}) \geqslant \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|}$.

Proof. We show this by contradiction. Let K be a τ-forbidden itemset for $\tau < 1$ and assume for the sake of contradiction that there exist subsets $I \subset J \subseteq K$ with $\mathsf{supp}(I,\mathcal{D}) - \mathsf{supp}(J,\mathcal{D}) < \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|}$. To simplify notation, we denote $\mathsf{supp}(I,\mathcal{D}) - \mathsf{supp}(J,\mathcal{D})$ by δ. We may then rewrite $\mathsf{supp}(K,\mathcal{D})$ as

$$\mathsf{supp}(K,\mathcal{D}) = \mathsf{supp}(K,\mathcal{D}) + \mathsf{supp}(I,\mathcal{D}) - \mathsf{supp}(J,\mathcal{D}) - \delta$$

Let L denote the subset $K \setminus (J \setminus I)$ of K. We have that $\mathsf{supp}(L,\mathcal{D}) = \mathsf{supp}(K,\mathcal{D}) - (\mathsf{supp}(J,\mathcal{D}) - \mathsf{supp}(I,\mathcal{D}))$, or equivalently that

$$\mathsf{supp}(L,\mathcal{D}) = \mathsf{supp}(K,\mathcal{D}) + \mathsf{supp}(I,\mathcal{D}) - \mathsf{supp}(J,\mathcal{D}).$$

Hence, $\mathsf{supp}(K,\mathcal{D}) = \mathsf{supp}(L,\mathcal{D}) - \delta$.

Furthermore, since $\mathsf{lift}(K,\mathcal{D}) \leqslant \tau$, Proposition 7 tells us that $L \subset K$ must satisfy $\mathsf{supp}(L,\mathcal{D}) \geqslant \frac{|\mathcal{D}|}{\sigma_I^{max} \times \tau}$. Knowing that $|\mathcal{D}| \geqslant \sigma_I^{max}$, we simplify the inequality to $\mathsf{supp}(L,\mathcal{D}) \geqslant \frac{1}{\tau}$. We show that this leads to a contradiction. Indeed, by definition we have that

$$\tau \geqslant \mathsf{lift}(K,\mathcal{D}) \geqslant \frac{|\mathcal{D}| \times \mathsf{supp}(K,\mathcal{D})}{\mathsf{supp}(K \setminus L,\mathcal{D}) \times \mathsf{supp}(L,\mathcal{D})}.$$

Since we can replace $\mathsf{supp}(K,\mathcal{D})$ by $\mathsf{supp}(L,\mathcal{D}) - \delta$:

$$\begin{aligned}\tau &\geqslant \frac{|\mathcal{D}|}{\mathsf{supp}(K \setminus L,\mathcal{D})} \times \frac{\mathsf{supp}(L,\mathcal{D}) - \delta}{\mathsf{supp}(L,\mathcal{D})} \\ &= \frac{|\mathcal{D}|}{\mathsf{supp}(K \setminus L,\mathcal{D})} \times \left(1 - \frac{\delta}{\mathsf{supp}(L,\mathcal{D})}\right),\end{aligned}$$

and by using that $\mathsf{supp}(L,\mathcal{D}) \geqslant \frac{1}{\tau}$ we have

$$\tau \geqslant \frac{|\mathcal{D}|}{\mathsf{supp}(K \setminus L,\mathcal{D})} \times \left(1 - \frac{\delta}{1/\tau}\right).$$

Since $\mathsf{supp}(K \setminus L,\mathcal{D}) \leqslant \sigma_K^{max} \leqslant \sigma_I^{max}$, we obtain that

$$\tau \geqslant \frac{|\mathcal{D}|}{\sigma_I^{max}} \times \left(1 - \frac{\delta}{1/\tau}\right).$$

From this, we can infer that $\tau \times \sigma_I^{max} \geqslant |\mathcal{D}| - (|\mathcal{D}| \times \delta \times \tau)$, and hence, $|\mathcal{D}| \times \delta \times \tau \geqslant |\mathcal{D}| - (\tau \times \sigma_I^{max})$. A further rearrangement of terms leads to

$$\delta \geqslant \frac{|\mathcal{D}|}{|\mathcal{D}| \times \tau} - \frac{\tau \times \sigma_I^{max}}{|\mathcal{D}| \times \tau} = \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|}.$$

Which gives a contradiction of our assumption that $\delta < \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|}$. As a consequence, the proposition holds. □

In particular, when applied to $I \subset J \subseteq K = J$, Proposition 8 indicates that for J to be τ-forbidden, we must have that $\mathsf{supp}(I,\mathcal{D}) - \mathsf{supp}(J,\mathcal{D}) \geqslant \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|}$ holds for any of its subsets I. During the depth-first search, when expanding I to J, if this condition is not satisfied, then J and all of its supersets K can be pruned.

Furthermore, it holds that $\frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|} > 0$, since $\sigma_I^{max} \leqslant |\mathcal{D}|$ and $\tau < 1$. Consequently, the proposition implies that τ-forbidden itemsets must have a support different from the supports of all their subsets, i.e., if J is τ-forbidden then $\mathsf{supp}(I,\mathcal{D}) > \mathsf{supp}(J,\mathcal{D})$ for any $I \subset J$. Itemsets J that satisfy this condition are called generators [86] or free itemsets [21]. A known property of generators is that all their subsets are generators as well; hence any subset of a τ-forbidden itemset is required to be a generator. If a non-generator itemset is encountered during the search, the entire subtree can therefore be pruned.

Our next pruning method uses the lift of an itemset to bound the support of its supersets. It follows from the observation that the *denominator* of the lift measure is in fact anti-monotonic.

Proposition 9. For any two itemsets I and J such that $I \subset J$, it holds that

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

Proof. We show the following: For any two itemsets I and J such that $I \subset J$, it holds that:

$$\min_{S \subset I}\big\{\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I \setminus S,\mathcal{D})\big\} \geqslant \min_{S \subset J}\big\{\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(J \setminus S,\mathcal{D})\big\}.$$

Clearly, this implies the proposition. We next show that the inequality holds. Indeed, this follows from the fact that any expression $\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I \setminus S,\mathcal{D})$ can be rewritten as

$$\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}((J \setminus S) \setminus X,\mathcal{D}),$$

where $X = J \setminus I$. Since

$$\mathsf{supp}((J \setminus S) \setminus X,\mathcal{D}) \geqslant \mathsf{supp}(J \setminus S,\mathcal{D}),$$

we obtain that the following inequality holds for any $S \subset I$,

$$\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I \setminus S,\mathcal{D}) \geqslant \mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(J \setminus S,\mathcal{D}),$$

as desired. □

Clearly, this lower bound can be used to prune itemsets J that cannot lead to τ-forbidden itemsets. When considering an itemset $I \subset J$ during our depth-first traversal, we already have the supports of subsets of I at our disposal. Hence, although $\mathsf{supp}(J,\mathcal{D})$ is not yet know, the value of the denominator is known when I is considered. This implies that we need a lower bound

on $\mathsf{supp}(J,\mathcal{D})$ to lower bound $\mathsf{lift}(J,\mathcal{D})$ using Proposition 9. We again use the trivial lower bound $\mathsf{supp}(J,\mathcal{D}) \geqslant 1$.

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

Proposition 10. If I is a τ-forbidden itemset then its frequency is bounded by $\mathsf{freq}(I,\mathcal{D}) \leqslant \frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$. Furthermore, for small τ-values this bound converges (from above) to $\frac{\tau}{4}$.

Proof. Consider an itemset I which is τ-forbidden for a given threshold $\tau < 1$. We first lower bound the lift of I by considering the maximal value that $\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I \setminus S,\mathcal{D})$ can take for $\emptyset \subset S \subset I$. Observe that $\mathsf{supp}(S,\mathcal{D}) + \mathsf{supp}(I \setminus S,\mathcal{D}) \leqslant |\mathcal{D}| + \mathsf{supp}(I,\mathcal{D})$. It can now be easily verified that the maximal value of the expression $\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I \setminus S,\mathcal{D})$ is obtained when $\mathsf{supp}(S,\mathcal{D}) = \frac{|\mathcal{D}|+\mathsf{supp}(I,\mathcal{D})}{2}$.

Since $\mathsf{lift}(I,\mathcal{D}) \leqslant \tau$ we then have that

$$\frac{\mathsf{supp}(I,\mathcal{D}) \times |\mathcal{D}|}{\left(\frac{|\mathcal{D}|+\mathsf{supp}(I,\mathcal{D})}{2}\right) \times \left(\frac{|\mathcal{D}|+\mathsf{supp}(I,\mathcal{D})}{2}\right)} \leqslant \mathsf{lift}(I,\mathcal{D}) \leqslant \tau.$$

We can use this to upper bound $\mathsf{supp}(I,\mathcal{D})$, as follows:

$$\mathsf{supp}(I,\mathcal{D}) \leqslant \frac{\tau}{4|\mathcal{D}|} \times \left(|\mathcal{D}|^2 + 2|\mathcal{D}| \times \mathsf{supp}(I,\mathcal{D}) + (\mathsf{supp}(I,\mathcal{D}))^2\right).$$

It is now a routine exercise to find the maximal value of $\mathsf{supp}(I,\mathcal{D})$. Indeed, one simply needs to solve the equation

$$\frac{\tau}{4|\mathcal{D}|} \times \left(|\mathcal{D}|^2 + 2|\mathcal{D}| \times \mathsf{supp}(I,\mathcal{D}) + (\mathsf{supp}(I,\mathcal{D}))^2\right) - \mathsf{supp}(I,\mathcal{D}) = 0.$$

We obtain the following two solutions:

$$\mathsf{supp}(I,\mathcal{D})^{\pm} = \frac{2|\mathcal{D}|}{\tau} - |\mathcal{D}| \pm 2|\mathcal{D}|\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}}.$$

Since $\frac{2|\mathcal{D}|}{\tau} - |\mathcal{D}| > |\mathcal{D}|$ for any $\tau < 1$, we see that $\mathsf{supp}(I,\mathcal{D})^+ = \frac{2|\mathcal{D}|}{\tau} - |\mathcal{D}| + 2|\mathcal{D}|\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} > |\mathcal{D}|$. This is impossible since no itemset can have a support strictly greater than $|\mathcal{D}|$. Hence, we are left with the other root $\mathsf{supp}(I,\mathcal{D})^- = \frac{2|\mathcal{D}|}{\tau} - |\mathcal{D}| - 2|\mathcal{D}|\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}}$. To obtain the upper bound on the frequency we divide by $|\mathcal{D}|$. This results in that the maximal frequency of a τ-forbidden itemset is $\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$.

To show the second statement in the proposition, we consider the Taylor expansion of $\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ given by

$$\frac{\tau}{4} + \frac{\tau^2}{8} + \frac{5\tau^3}{64} + \frac{7\tau^4}{128} + O(\tau^5).$$

Hence, for decreasing values of τ, the maximal frequency of a τ-forbidden itemset converges from above to $\tau/4$. □

The proposition tells that for small values of τ, τ-forbidden itemsets are at most approximately $\tau/4$-frequent, and that itemsets whose *frequency* exceeds $\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ cannot be τ-forbidden.

This result can also be used to obtain an initial estimate for τ. Clearly, $\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ should be greater than 0, or no itemsets will be forbidden. When picking a higher τ, the bound indicates how frequent forbidden itemsets can get. If this possible frequency gets substantially high, the chosen value τ is most probably too high. The actual value of $\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ then gives a useful indication of how frequent the discovered forbidden itemsets will be.

6.4.2 Mining Forbidden Itemsets

We now present an algorithm, FBIMINER, for mining τ-forbidden itemsets in a dataset $\mathcal{D}$. That is, the algorithm computes $\mathsf{dirty}(\mathcal{D}, \Gamma, q)$ for the language Γ and predicate q defined earlier. The algorithm is based on the well-known Eclat algorithm for frequent itemset mining [116]. Here, we only describe the main differences with Eclat. The pseudo-code of FBIMINER is shown in Algorithm 10.

The algorithm takes as input a projected dataset in vertical layout $\mathcal{D}\downarrow[P]$, consisting of all objects which contain a prefix itemset P, and the lift threshold τ. The initial call uses the entire dataset $\mathcal{D}$ in vertical data layout, and an empty prefix itemset $P = \emptyset$. Just like Eclat, FBIMINER employs a depth-first search of the itemset lattice (**for** loop line 3 – 25, and recursive call on line 25). When expanding an itemset I in the search tree (line 4), it is processed (line 5 – 16), before the next layer of the search tree is generated (line 17 – 24). New itemsets are generated by extending I with all items in the dataset that occur in the objects in I's cover (line 18 – 24). To efficiently compute the support of the new itemsets, Eclat uses set intersections of the covers of the items i and j (line 19). If the size of the newly computed cover C is higher than zero, i.e., the new itemset $J = I \cup \{j\}$ is supported, it is added to $\mathcal{D}\downarrow[I]$, the dataset $\mathcal{D}$ in vertical layout projected on I. When generating new itemsets, the candidate items j are added according to a total ordering on the items, i.e., items are only added when they come after each item in I (line 18). Items are ordered by ascending support, as this is known to improve efficiency. More details about efficient implementation strategies for Eclat are discussed in [12].

A first challenge is to tweak the Eclat algorithm such that the lift of itemsets can be computed. Observe that the lift of an itemset is dependent on the support of *all* of its subsets. For this purpose, we use the same depth-first traversal as Eclat, but traverse it in reverse order (line 3). Indeed, such a reverse pre-order traversal of the search space visits all subsets of an itemset I before visiting I itself [25]. This is exactly what is required to compute the lift measure, provided that the support of each processed itemset is stored.

Algorithm 10 Mining low lift τ-forbidden itemsets

1: **procedure** FBIMINER($\mathcal{D}\downarrow[P], P \subseteq \mathcal{I}, \tau$)
2: $\mathsf{FBI} \leftarrow \emptyset$
3: **for all** $i \in \mathcal{I}$ occurring in $\mathcal{D}\downarrow[P]$ in reverse order **do**
4: $I \leftarrow P \cup \{i\}$
5: **if** not ISGENERATOR(I) **then**
6: **continue**
7: STOREGENERATOR(I)
8: **if** $|I| > 1$ & $\mathsf{freq}(I,\mathcal{D}) \leqslant \frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ **then**
9: **if** a subset of I has been pruned **then**
10: **continue**
11: **if** $\mathsf{lift}(I,\mathcal{D}) \leqslant \tau$ **then**
12: $\mathsf{FBI} \leftarrow \mathsf{FBI} \cup \{I\}$
13: **if** $\frac{|\mathcal{D}|}{\tau} > \min_{S \subset I} \{\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I \setminus S,\mathcal{D})\}$ **then**
14: **continue**
15: **if** $\mathsf{supp}(I,\mathcal{D}) < \frac{|\mathcal{D}|}{\sigma_I^{max} \times \tau}$ **then**
16: **continue**
17: $\mathcal{D}\downarrow[I] \leftarrow \emptyset$
18: **for all** $j \in \mathcal{I}$ in $\mathcal{D}$ such that $j > i$ **do**
19: $C \leftarrow \mathsf{cov}(\{i\},\mathcal{D}) \cap \mathsf{cov}(\{j\},\mathcal{D})$
20: $J \leftarrow I \cup \{j\}$
21: $\mathsf{supp}(J,\mathcal{D}) \leftarrow |C|$
22: **if** $\mathsf{supp}(I,\mathcal{D}) - \mathsf{supp}(J,\mathcal{D}) \geqslant \frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|}$ **then**
23: **if** $\mathsf{supp}(J,\mathcal{D}) > 0$ **then**
24: $\mathcal{D}\downarrow[I] \leftarrow \mathcal{D}\downarrow[I] \cup \{(j,C)\}$
25: $\mathsf{FBI} \leftarrow \mathsf{FBI} \cup$ FBIMINER($\mathcal{D}\downarrow[I], I, \tau$)
26: **return** FBI

However, Eclat generates a candidate itemset based on only two of its subsets [116]. Hence, the supports of all subsets of an itemset are not immediately available in the search tree. To remedy this, we store the support of the processed itemsets using a prefix tree, for time- and memory-efficient lookup during lift computation.

Having integrated lift computation in the algorithm, we next turn to our pruning and optimization strategies. We deploy four pruning strategies (lines 9, 13, 15 and 22). The first strategy (line 9) applies to itemsets I for which the lift cannot be computed, which happens when some of its subsets are pruned away in an earlier step. The absence of subsets is detected when the lift computation requests the support of a subset that is not stored. Recall that we derived pruning properties of the lift measure, that must hold for all subsets of a forbidden itemset. If a subset of I was pruned because of these properties, the subset did not satisfy the required property. This implies that I (and all its supersets J) cannot be τ-forbidden and thus can be pruned (line 10).

The second pruning strategy (line 13) applies to supersets of itemsets I for

which we have been able to compute their lift, and leverages Proposition 9. Indeed, if we have that $\frac{|\mathcal{D}|}{\tau} > \min_{S \subset I}\{\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I \setminus S,\mathcal{D})\}$ then Proposition 9 tells that I cannot be a subset of a τ-forbidden itemset. Hence, all itemsets in the tree below I are pruned (line 14).

By contrast, the third strategy (line 15) leverages Proposition 7 and skips supersets of itemsets I, regardless of whether their lift was computed. Indeed, if $\mathsf{supp}(I,\mathcal{D}) < \frac{|\mathcal{D}|}{\sigma_I^{max} \times \tau}$ holds, then I cannot be part of a τ-forbidden itemset, resulting in a further pruning of the search space.

The fourth strategy employs Proposition 8 to prune extensions of I that do not cause a sufficient reduction in support. When generating direct supersets J of I, we have immediate access to their supports. If the reduction in support is smaller than $\frac{1}{\tau} - \frac{\sigma_I^{max}}{|\mathcal{D}|}$, the proposition implies that J (and its supersets) cannot be τ-forbidden, hence we do not need to add it to the next layer of the search tree. This check is performed prior to the recursive call (line 22).

Finally, we also implement an optimization that avoids certain lift computations (line 8). Only when the algorithm encounters an itemset I with at least two items and a frequency lower than the bound from Proposition 10, the lift of I is computed. All other itemsets cannot be τ-forbidden by Proposition 10. Note that this only eliminates the need for checking the lift of certain itemsets but by itself does not lead to a pruning of its supersets.

A careful reader may have spotted the optimized pruning of non-generators on lines 5-7. Recall that as a direct consequence of Proposition 8, any τ-forbidden itemset must be a generator, i.e., have a support which is strictly lower than that of all of its subsets. The Talky-G algorithm [101] implements specific optimizations for mining such generators, using a hash-based method that was introduced in the Charm algorithm for closed itemset mining [114]. We use the same technique in FBIMINER to efficiently prune non-generators.

More specifically, during the mining process, all encountered generators are stored in a hashmap (procedure STOREGENERATOR on line 7). As hash value we use, just like the Charm algorithm, the sum of the tid's of all objects in which an itemset occurs. If an itemset has the same support as one of its subsets, it is clear that this itemset must occur in exactly the same objects, and will map to the same hash value. Moreover, the probability of unrelated itemsets having the same *sum of oids* is lower than the probability of them having the same support. Therefore this sum is taken as hash value instead of the support of itemsets.

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

We conclude this section by verifying the correctness of FBIMINER.

Proposition 11. Let $\mathcal{D}$ be a dataset and τ a maximum lift threshold. Algorithm FBIMINER$(\mathcal{D}\downarrow[\emptyset], \emptyset, \tau)$ returns all τ-forbidden itemsets in $\mathcal{D}$.

Proof. Consider FBIMINER from which all pruning steps are removed. In this case, the algorithm will perform an exhaustive depth-first search traversal of all itemsets and only (and all) τ-forbidden itemsets will be added to FBI (lines 11 and 12). In other words, the algorithm correctly computes all τ-forbidden itemsets in $\mathcal{D}$.

With pruning and optimizations enabled, suppose that for the sake of contradiction, there is a τ-forbidden itemsets I that is not returned by FBIMINER. This implies that I was not added to FBI (lines 11 and 12). There are two possible causes for this: (i) the condition $\mathsf{freq}(I,\mathcal{D}) \leqslant \frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ (line 8) is not satisfied; or (ii) a subset J of I has been pruned. Clearly, case (i) cannot happen. Indeed Proposition 10 implies that $\mathsf{lift}(I,\mathcal{D}) > \tau$, a contradiction. Similarly, case (ii) leads to a contradiction. When a subset J of I is pruned, Propositions 7, 8 and 9 imply that all supersets of J, including I, must have a lift greater than τ. We may thus conclude that FBIMINER is correct. □

6.5 User Interaction

Our FBI cleaning method has been designed such that it can be run fully automatically, without any user input or interaction. While it is desirable to alleviate the user of any such obligations, of course, in practice, we wish to *allow* the user to control the algorithms to a certain degree, if desired.

Filtering Forbidden Itemsets. Recall the forbidden itemsets discovered in the Zoo dataset, from Table 6.1. The platypus may be a prime candidate for containing illegal value combinations, and is certainly suspicious, but it is not an actual error. An eventual user should be able to prevent the platypus from being "repaired". Such a mechanism of user interaction can readily be integrated into our method. As we will explain in-depth in the next chapter, the repairing process relies only on the set of forbidden itemsets, $\mathsf{FBI}(\mathcal{D},\tau)$, and another set $\mathbb{A}$ of itemsets at risk of becoming τ-forbidden. Regardless of how these sets are obtained, the repair algorithm attempts to remove the itemsets in $\mathsf{FBI}(\mathcal{D},\tau)$ from the data while guaranteeing that no sets in $\mathbb{A}$ become τ-forbidden. This makes it straightforward for the user to forcibly allow certain itemsets, that would normally be considered forbidden. Indeed, the FBIMINER algorithm could present the user with a set $\mathsf{FBI}(\mathcal{D},\tau)$ of itemsets it deems forbidden, and before the repairing starts, the user can then decide which of these itemsets should effectively be removed from $\mathcal{D}$ by the repairing process (if not all of them). Any itemset which the user does not want to remove, simply has to be discarded from the set $\mathsf{FBI}(\mathcal{D},\tau)$. Returning to the Zoo example, the itemsets related to the platypus and scorpion could be filtered out of $\mathsf{FBI}(\mathcal{D},\tau)$ by a user, leaving $\mathsf{FBI}(\text{Zoo}, 0.1) = \{(\text{AQUATIC=0}, \text{BREATHES=0 (CLAM)})\}$. Clearly, any itemset that the user removes from $\mathsf{FBI}(\mathcal{D},\tau)$, should automatically be removed from $\mathbb{A}$ as well. Moreover, the

set $\mathbb{A}$ can be filtered in a similar fashion, removing itemsets that the user "allows to become forbidden", i.e., their lift may drop below τ. Consequently, the form of user interaction outlined above has an impact on the problem statement. The output of our cleaning method is no longer a database that is clean by our definition, but is instead clean by the standards of the user, who has "allowed" some forbidden itemsets.

Choosing The Best Modifications. Besides filtering the itemsets to be removed from the dataset, the user can also make direct decisions on the repairs. Instead of simply repairing using the first safe modification encountered, our repair algorithm may find a number of possible repairs and display them to the user, of course guaranteeing safety. The user then decides which of the suggested modifications makes the most sense semantically, and the algorithm performs this modification.

Iteratively Determining τ. Another possible advantage of user interaction pertains to setting the maximum lift threshold τ. Instead of specifying τ upfront, it can also be determined empirically through iterative user interaction. FBIMINER can be adapted to return the top-k lowest lift forbidden itemsets, which may then be filtered by the user, as described earlier. The itemsets in $\mathsf{FBI}_{top\text{-}k}(\mathcal{D})$ are then removed from the dataset during repairing. The entire process is then repeated, returning a new set of top-k forbidden itemsets. This is repeated until the user filters all itemsets in $\mathsf{FBI}_{top\text{-}k}(\mathcal{D})$, i.e., none of the top-$k$ lowest lift itemsets are actually considered errors.

To discover such a top-k of lowest lift itemsets, FBIMINER needs to be initialized with a high (but allowed) value of τ, i.e., $\tau \approx 3/4$. During the search, the current k lowest lift itemsets are kept in a list TOP-k. Once k itemsets have been processed, the value τ is continuously lowered to the highest lift value in TOP-k, in order to maintain maximal pruning capabilities. Whenever a new itemset with lift $< \tau$ is encountered, it replaces the itemset in TOP-k with the highest lift, and τ is lowered again. Clearly, when no more itemsets with lift $< \tau$ can be found, the list TOP-k contains the k itemsets with lowest lift in the dataset.

6.6 Related Work

We have already discussed some of the most related constraint formalisms in Section 6.3.2, where we made the case that most real-world scenarios only demand for formalisms that concern violations consisting of single objects. In the following section, we provide more insight into the relations between these formalisms and forbidden itemsets. Moreover, the concept of forbidden itemsets draws heavily upon advances made in the area of data mining and knowledge discovery. Exploratory data mining methods are typically used to discover patterns which are somehow interesting and surprising [108]. Depending on the interestingness measure, as with low lift, those patterns can be considered errors. We also discuss related work in these areas. There has been a vast amount of research in this field; we only consider work related to error detection. Repairing has received little attention in the data mining community.

6.6.1 Edits

In terms of constraint formalisms, our notion of forbidden itemsets is closest in spirit to edits, introduced in Herzog et al. [59] and Fellegi and Holt [48], widely used by census bureaus. Edits specify sets of illegal value combinations on a single object. More formally, an edit η over $\mathcal{D}$ is a Cartesian product over subsets of all domains of $\mathcal{A}$, denoted as $\eta = \times_{A_i \in \mathcal{A}} S_i$ where each $S_i \subseteq \mathsf{dom}(A_i)$. An object o is said to be dirty if $o \in \eta$. It is readily verified that an edit can be seen as a union of forbidden itemsets, and vice versa, a forbidden itemset corresponds to an edit. For example, the edit $\mathsf{dom}(Sex) \times \{\text{Age=} \leq 12, \text{Age=} \geq 80\} \times \{\text{Employed=True}\}$ corresponds to the two forbidden itemsets (AGE=$\leq$ 12, EMPLOYED=TRUE) and (AGE=$\geq$80 , EMPLOYED=TRUE). Edits thus capture the class of inconsistencies when no condition on low lift is imposed. To our knowledge, no prior work has addressed the issue of discovering edits from dirty data. Indeed, in census studies edits are typically provided by domain experts. Furthermore, since they are not discovered by algorithms, edits are typically not associated with any quality measure such as lift. However, our repairing algorithm draws inspiration from edit-based repairing of census data. Repairs using edits often make use of so-called hot deck imputation [59], which is very similar to the nearest neighbour method we use to obtain candidate modification. However, in the context of edits known up front, guaranteeing that no violations occur in the data after repairs is a very different challenge from our dynamic notion of cleanliness.

6.6.2 Constraint Discovery and Data Repair

In general, constraints may be expressed on the attribute level, with violations spanning across multiples objects. On the other hand, forbidden itemsets only consider value combinations within a single object. Most relevant to our work are therefore the formalisms whose violations are confined to single objects, such as constant CFDs [46] and constant Denial Constraints [31]. Algorithms such as those presented in Chapter 4 [93], Chiang and Miller [29], Chu et al. [33], and Fan et al. [46], can discover these constraints from clean or moderately dirty data. With a set of constraints in place, the next step is to modify the data such that the constraints are satisfied. Alternatively, some repair methods, such as Chiang and Miller [30], Beskales et al. [16], and Mazuran et al. [78], consider modifying some of the constraints in order to obtain a clean database. More recent work in Rezig et al. [98] considers the impact of repairs on the constant patterns pertaining to a given set of FDs.

The main distinction with our FBI cleaning system, is that none of the techniques cited above consider constraint discovery and repairing in unison. Indeed, after a separate constraint discovery step, the repair algorithm judges cleanliness solely on the basis of the previously discovered constraints. It is then not guaranteed, and typically not the case, that re-running the discovery algorithm after repairing yields a "clean" database, i.e., no more violations are found. In fact, the number of violations discovered

might be greater than before repairing. By contrast, we have adopted the dynamic notion of dirtiness in which the considered constraints depend on the data and vary when modifications to the data are made. As we will show in the next chapter, our repair algorithm is closely interwoven with forbidden itemset mining, in order to ascertain the cleanliness of our output. Only by considering constraint discovery and repair in unison, is it possible for a repair method to guarantee cleanliness in terms of the discovery algorithm. To our knowledge, no prior work has addressed the issue of repairing when new constraints may arise due to the repair itself. As such, our repair method, presented in the next chapter, faces new challenges that differ substantially from existing work.

6.6.3 User Interaction

Aside from human experts providing constraints, other forms of user input for data cleaning have been considered. Crowdsourced data cleaning is mostly focused on entity resolution, as in Wang et al. [105], in contrast to semantic errors. Since semantic errors are usually data-dependent, it is difficult to assess such errors without context. Recent work has considered user interaction in the form of algorithms which are guided by the user, such as Volkovs et al. [103], He et al. [58], Thirumuruganathan et al. [102], and our method from the previous chapter Rammelaere and Geerts [94]. Such algorithms present preliminary results to be evaluated by a user, and employ this feedback in order to obtain more preferred results. Other methods, such as Yakout et al. [110], consider user interaction in the context of selecting repairs. Our method naturally lends itself for user interaction. The effect of such interaction, however, is beyond the scope of this dissertation.

6.6.4 Association Rules

An association rule is of the form $I \to J$ where I and J are two (disjoint) itemsets. Typically, association rules of interest are those with high support, and high confidence, although many interestingness measures exist. One may regard an object o to be dirty if it supports I but does not support J. We further motivate our use of lift instead of confidence experimentally, in Section 6.7.1. The relationship between association rules and forbidden itemsets was described in Section 6.3.1, where we defined the lift of an itemset in terms of the lift of its embedded rules. Indeed, an itemset K is τ-forbidden if there is a low lift rule $I \to J$ between every partitioning of K into disjoint subsets I and J. We can thus regard a forbidden itemset as a collection of low lift rules.

However, by considering forbidden itemsets rather than rules, we prevent ambiguity when interpreting errors due to the directionality of rules. For example, if the rule $X \to A$ is found to be unlikely, but the rule $A \to X$ is not, should the combination of X and A in an object really be an error?

Furthermore, while "interesting" rules are typically defined based on a *high* confidence or lift, forbidden itemsets specifically demand for low lift. This is because we are directly interested in the violations of rules, since, there are many violations which cannot be expressed as a single, positive

association rule. For example, if people of young age cannot have a driver's license, then $Age = 12$ is clearly an illegal value. However, there can be no association rule stating which exact value the Age attribute *should* have, since they cannot express $Age \geq 16$. Indeed, one would require a specific rule for each value of the Age attribute ≥ 16; and it is unlikely that any one rule $License = 1 \rightarrow Age = X$ will have a high confidence or lift.

6.6.5 Anomaly and Outlier Detection

The detection of semantic inconsistencies in data is related to anomaly and outlier detection (for recent surveys, see Aggarwal [4], Chandola et al. [27], or Markou and Singh [77]). Most work in these areas focuses on continuous data, where datapoints are considered dirty if they are distant from other points or if they are statistical outliers. Such methods are often not applicable to our setting, which is focused on categorical data, and where a dirty and a clean object could possibly differ in only one value. Outliers found with statistical methods for categorical outlier detection, such as dBoost from Pit-Claudel et al. [88], can overlap with forbidden itemsets, but require careful tuning of parameters, as noted in Abedjan et al. [2]. Finally, the repair problem has received little attention in the field of outlier detection, where dirty data points are usually removed instead of repaired. Indeed, outliers are not declarative, and hence it is difficult to reason about the prevention of outliers after repairing, which is the premise of this chapter.

The anomaly detection method presented in Bertens et al. [14] uses a measure very similar to lift, in their definition of "Class 2 Anomalies". These anomalies show considerable overlap with the forbidden itemsets. However, the goal of this method differs substantially as the exploratory method focuses on efficiently discovering the most interesting patterns, whereas we are interested in finding *all* forbidden itemsets *that satisfy* a hard threshold in order to repair them. For the sake of efficiency, the method in Bertens et al. [14] does not consider all subsets of an itemset I to measure the interestingness of I. Instead, a set of descriptive patterns S is computed up front, and measures the interestingness of I is based only on those $s \in S$ with $s \subset I$. From this, it is clear that the method is ill-suited to a setting of dynamic data cleaning, since even the patterns on which the lift is computed would change after repairing.

6.6.6 Mining in Dirty Data

Another area of data mining research is concerned with discovering patterns in dirty databases, often called Error-Tolerant Itemsets (ETI), as in Pei et al. [87] and Gupta et al. [56]. Such methods discover frequent patterns, but do not require that a pattern *exactly* occurs in its supporting transactions. In an error detection context, one could interpret a frequent ETI as an itemset which should occur in a set of transactions, althought it doesn't exactly occur in all of them. This is conceptually inverse to the forbidden itemsets we propose. As with outlier detection and exploratory data mining, repairing ETI's has not been thoroughly investigated.

Table 6.2: Statistics of the UCI datasets used in the experiments. We report the number of objects, distinct items, and attributes.

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$
Adult	48842	202	11
CensusIncome	199524	235	12
CreditCard	30000	216	12
Ipums	70187	364	32
LetterRecognition	20000	282	17
Mushroom	8124	119	23
Soccer	200000	10	767

6.7 Experiments

The experiments were conducted on real-life datasets from the UCI repository [39]. We show results for six datasets. Additionally, we evaluate the precision of our error detection on the synthetic Soccer dataset [1]. The relevant statistics of these datasets are given in Table 6.2. The Adult database was preprocessed by discretizing ages and removing other continuous attributes. The algorithms have been implemented in C++, the source code and used datasets are available for research purposes [91]. The program has been tested on an Intel Xeon Processor (2.9GHZ) with 32GB of memory running Linux. Our algorithms run in main memory. Reported runtimes are always an average over five independent runs.

6.7.1 Likeliness Function

In sections 6.3 and 6.6, we have already provided some intuition behind the choice for low lift itemsets as an error detection method. The low lift forbidden itemsets rely on two "design choices". Firstly, we have opted to use lift instead of, for instance, confidence, as commonly used in approximate CFDs or association rules. Secondly, we have opted to define the lift of an itemset using two partitions, i.e., based on the association rules that can be created by splitting the itemset in two. We next provide examples and results to support our design choices and the general usefulness of forbidden itemsets to detect errors.

We have opted for the lift measure as opposed to confidence, since confidence does not take correlation between items into account. Here, we define confidence of an itemset as the highest confidence of any association rule between two partitions of the itemset, similar to how we defined lift. For example, on the Adult dataset, the itemset (MARITALSTATUS=WIDOWED, COUNTRY=GUATEMALA) can be discovered with a confidence of 0.05. Clearly, widowed people in Guatemala are not erroneous; this itemset is simply a consequence of the low frequency of the item (COUNTRY=GUATEMALA) in

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

Table 6.3: Five least likely itemsets discovered in Adult dataset, using different likeliness functions

Likeliness	**Example Itemsets**
Pairwise Lift	Sex=Female, Relation=Husband Sex=Male, Relation=Wife Relation=Not-in-family, Marital=Married Marital-status=Married, Age=<18 Relation=Husband, Age=<18
Full Indep.	Sex=Female, Relation=Husband Sex=Male, Relation=Wife Relation=Not-in-family, Marital=Married, Educ.=College Relation=Not-in-family, Marital=Married,Occup.=Other Relation=Not-in-family, Marital=Married, Age=> 50
Confidence	Sex=Female, Relation=Husband Country=Mexico, Race=Asian-Pac-Islander Age=18-21, Education=Masters Occupation=Protective-serv, Educ.=Prof-school Occupation=Machine-op-inspct, Educ.=Prof-school

this dataset. In fact, a similar low-confidence itemset can be found with other Country-values, such as Honduras. When using the lift measure, however, the lift of (MARITALSTATUS=WIDOWED, COUNTRY=GUATEMALA) now becomes 1.46, indicating that these items in fact have a positive correlation in the Adult dataset. As such, a co-occurence of this items is unlikely to be an inconsistency.

Our definition of the lift on an itemset uses a bipartitional model. One might also define the lift using an arbitrary number of partitions, up until the number of items in the itemset. In fact, such a "full independence" assumption has been used before to define lift in Zaki and Meira Jr [115, p. 310]:

$$\mathsf{lift}_1(I,\mathcal{D}) := \frac{\mathsf{freq}(I,\mathcal{D})}{\mathsf{freq}(\{i_1\},\mathcal{D}) \times \cdots \times \mathsf{freq}(\{i_k\},\mathcal{D})}.$$

However, this definition introduces an undesirable bias towards larger itemsets: many items with a slight negative correlation might have a lower lift than two items with a strong negative correlation, as the expected frequency does not take the number of items into account. In other words, by using information from all subsets of the itemset, we increase the robustness and reduce the false positive rate of our measure, with the aim of attaining a high precision. The advantages of the bipartitional model are further supported by the theoretical analysis performed in [37].

We illustrate the outlined issues with confidence and full independence lift in Table 6.3. This table displays the five least-likely itemsets using the

Table 6.4: Precision of discovered forbidden itemsets on Adult dataset.

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

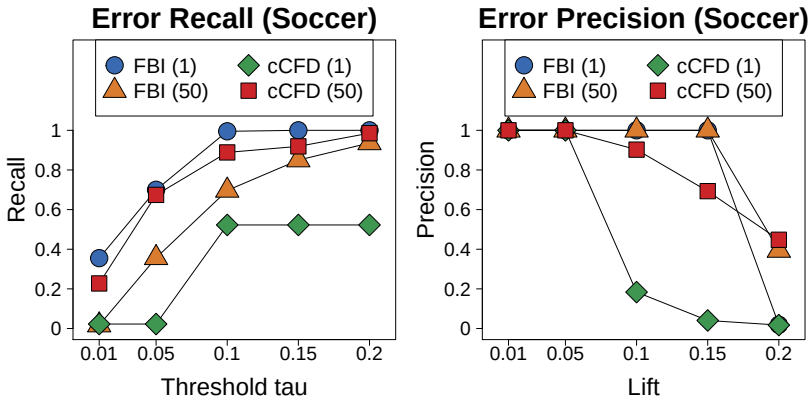


Figure 6.2: Precision and recall of dirty objects found by Forbidden Itemsets and approximate cCFDs, for errors generated by 1 and 50 cCFDs, in function of threshold τ.

different measures, where "pairwise lift" is the lift measure used throughout this chapter for forbidden itemsets. All three measures agree that the itemset (SEX=FEMALE, RELATION=HUSBAND) is the most probable error in the dataset. Full independence lift's bias towards larger itemsets shows with respect to the itemset (RELATION=NOT-IN-FAMILY, MARITAL=MARRIED): pairwise lift detects this itemset as one of the least likely itemsets. Full independence lift, however, ranks multiple supersets of this problematic itemset as *less likely*. The addition of items, such as Age=$\geqslant$50, for example, does not add much information, since the core problem is the co-occurence of Relation=Not-in-family and Marital=Married. When considering confidence instead of lift, we see the expected problems with items that are simply uncommon, but not necessarily correlated. For example, (AGE=18-21, EDUCATION=MASTERS) would not be very unlikely according to lift, and is probably not an error (people born at the end of the year doing a one-year masters are 21 when they finish), but scores very low in terms of confidence because of the small number of people in the data with a Masters education.

6.7.2 Effectiveness of Forbidden Itemsets

Next, we turn our attention towards the effectiveness of our Forbidden Itemsets formalism. An intuitive question to ask is, "Are the forbidden itemsets actually errors?". To provide a definitive answer to this question, a gold standard for the subjective task of data cleaning would be needed. However,

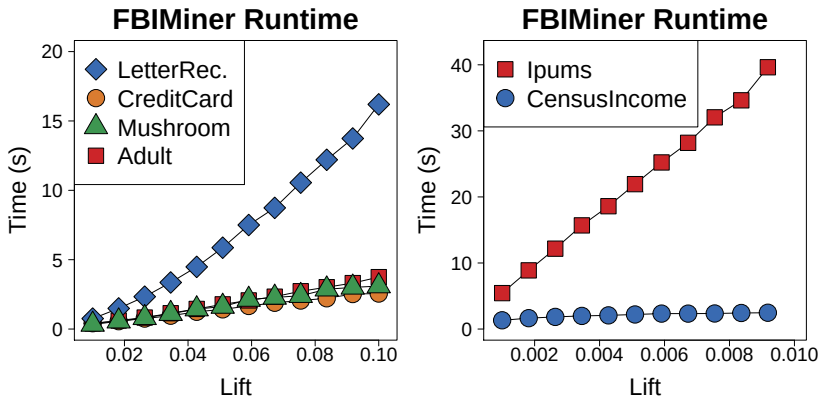


Figure 6.3: Runtime of FBIMiner in function of maximum lift threshold τ.

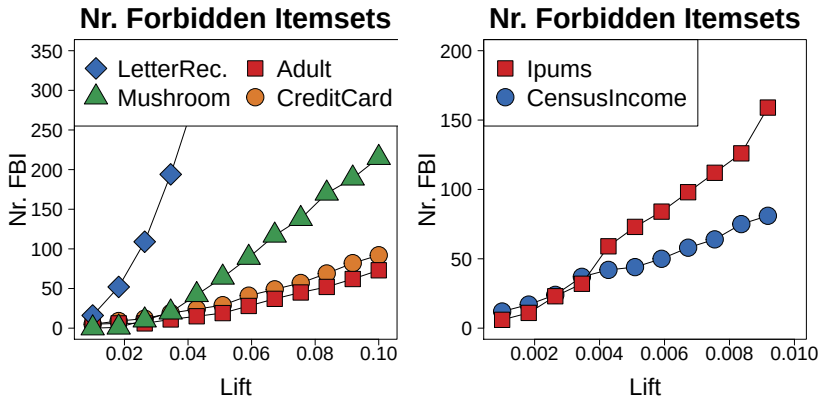


Figure 6.4: Number of Forbidden Itemsets in function of maximum lift threshold τ.

such gold standard datasets are not readily available. In line with Chiang and Miller [29], we have therefore evaluated forbidden itemsets manually for usefulness, obtaining only a precision score. Computing recall is impossible in our context: we have no way to know what all the errors in these datasets are. The precision results for the Adult dataset, which is the most readily interpretable, are shown in Table 6.4. We see that the precision is very high for small τ-values, and keeps up around 50% in the entire τ-range, which is quite high for an uninformed method. We believe such a high precision is important to instil faith in an eventual user; if the majority of the inconsistencies we discover are not actual errors, then a user is unlikely to believe that our cleaning method works. On the other hand, a low recall might be less noticeable, and indeed less detrimental to the outcome: successfully cleaning a part of all the errors in a dataset is still a good result.

In order to obtain a more systematic evaluation of precision than just manual inspection, we perform an additional experiment by generating errors for the synthetic Soccer dataset. We compare our precision results to a

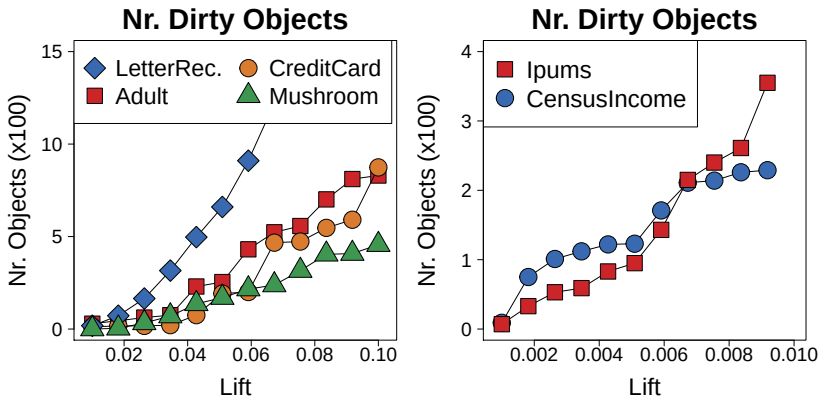


Figure 6.5: Number of objects containing one or more Forbidden Itemsets in function of maximum lift threshold τ.

similar constraint formalism, namely approximate constant CFDs (cCFDs). To introduce errors in the data, we use the BART [8] error generator. However, BART requires the constraints to be known up front, and is hence not directly applicable to our method. We have therefore first mined cCFDs on the Soccer dataset (which is known to be clean), with a confidence of 100% and a minimum support of 2000. Subsequently, BART was used to insert violations pertaining to a set number of randomly selected cCFDs, from this set of 100% confident cCFDs. In Figure 6.2, we show the precision and recall of detected dirty objects for increasing values of τ, for both forbidden itemsets with maximum lift τ, and approximate cCFDs with confidence $1-\tau$.

Here, we see that forbidden itemsets are able to detect more dirty objects while retaining a perfect precision than cCFDs. It is important to note that all errors were, in theory, detectable using cCFDs, since they were generated from cCFDs. As discussed before, forbidden itemsets in general can capture different types of errors as well. In the case of violations pertaining to a single cCFD, forbidden itemsets can reliably discover all dirty objects on the synthetic data, reinforcing the point that forbidden itemsets are very well-suited for data with a low error rate.

6.7.3 Discovering Forbidden Itemsets

So far, we have focused on the usefulness of our forbidden itemsets formalism. We next investigate the performance of our algorithm for discovering forbidden itemsets, FBIMINER. For this experiment, we ran FBIMINER with full pruning. We report the total runtime, the number of forbidden itemsets, and the number of objects containing a forbidden itemset, for increasing values of τ. For the larger datasets, Ipums and CensusIncome, a smaller τ range was considered. This prevents an explosion in the number of forbidden itemsets and the associated high runtime. The results are shown in Figure 6.3, Figure 6.4 and Figure 6.5, respectively.

The results show that the runtime of the algorithm (Figure 6.3) scales well

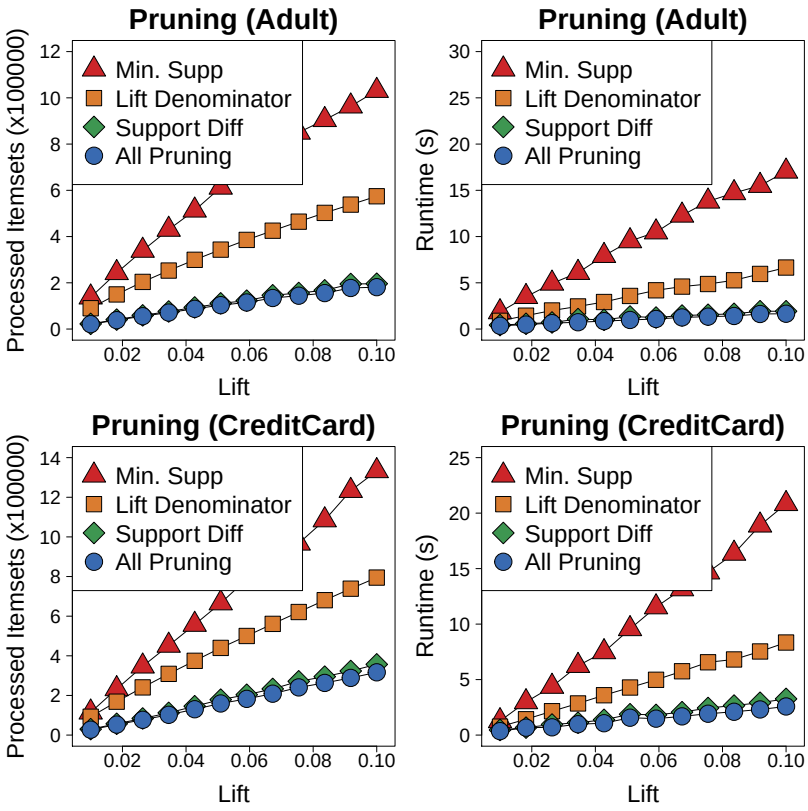


Figure 6.6: Runtime and number of itemsets processed with different pruning strategies in function of maximum lift threshold τ.

with τ. As a result of the depth-first search, the runtime is strongly influenced by the number of distinct items. As a consequence, the algorithm runs slowest on the Ipums dataset. The runtime on the LetterRecognition dataset is explained by its relatively high number of items, and the simple fact that it contains many forbidden itemsets. Indeed, it is clear that the "regularity" of the data also impacts our runtimes: if there are few negative correlations between items, our pruning strategies will quickly discern the fact that no itemsets can be forbidden.

The number of forbidden itemsets (Figure 6.4) is typically small, although there is a stronger than linear increase as the lift threshold increases, illustrating that τ should indeed be chosen very small. Especially for the LetterRecognition database, the number of forbidden itemsets increases exponentially. This is because the dataset is very noisy, since the contained letters were randomly distorted. In contrast, the less noisy Adult and CensusIncome datasets have relatively few dirty objects. The number of dirty objects (Figure 6.5) naturally follows a similar pattern as the number of forbidden itemsets, with an occasionally big increase if a forbidden itemset with a relatively high support is discovered.

Table 6.5: Runtime influence of maximal frequency bound in function of τ. We show the percentage decrease/increase in runtine when using the frequency bound, as opposed to the runtime without frequency bound.

	τ-value					
Dataset	0.01	0.026	0.043	0.067	0.084	0.1
Influence on runtime						
Adult	-8%	-18%	-8%	-7%	-2%	-2%
CensusIncome	+16%	+23%	+25%	+28%	+30%	+31%
CreditCard	-8%	-18%	-10%	+1%	+1%	+3%
Ipums	+37%	+50%	+54%	+58%	+62%	+62%
LetterRecognition	-3%	+1%	0%	-2%	+2%	+1%
Mushroom	-15%	-33%	-28%	-14%	-12%	-8%
Influence on number of visited nodes						
Adult	+4%	0%	0%	0%	0%	0%
CensusIncome	+23%	+33%	+38%	+41%	+38%	+37%
CreditCard	+5%	0%	0%	0%	0%	0%
Ipums	+38%	+49%	+56%	+61%	+63%	+64%
LetterRecognition	+1%	+1%	0%	0%	0%	0%
Mushroom	+5%	+6%	+1%	0%	0%	0%

6.7.4 Pruning Properties

The runtimes of FBIMINER were obtained with all pruning enabled. We now wish to verify that each of the different strategies indeed contributes to lower runtimes. In order to evaluate the influence of the pruning strategies, we report the number of itemsets processed with only one type of pruning enabled, and contrast these with the number of itemsets processed when all pruning is enabled. We also show the resulting runtimes. We distinguish between *Min. Supp* pruning, using Proposition 7 on line 15 of Algorithm 10; *Lift Denominator* pruning, using Proposition 9 on line 13; and *Support Diff* pruning, using Proposition 8 on line 22.

The results are shown in Figure 6.6 for the Adult and CreditCard datasets; results for CensusIncome and LetterRecognition were similar. On the Mushroom and Ipums datasets, which have many attributes, FBIMiner became infeasible for larger values of τ without Support Diff pruning. Clearly, Support Diff pruning is dominant in most cases. Since this strategy also entails

non-generator pruning, it is definitely crucial for the runtime of FBIMiner. Especially as τ increases, the other strategies also improve the overall result, indicating that all are beneficial and complementary to each other. We do not show the results when all pruning is disabled since all itemsets are then considered (independently of τ), leading to a high number of processed itemsets and running time.

To conclude this experimental section, we address the bound on the maximal frequency of a forbidden itemset, used on line 8 of Algorithm 10. Recall that this is not a pruning strategy: using the frequency bound effectively increases the number of itemsets processed, since it disables the pruning strategies that require the lift to be computed first. On the other hand, the frequency bound may reduce runtime by avoiding certain unnecessary lift computations. This bound was disabled for the previous pruning results, to prevent painting a distorted picture of the influence of each pruning strategy. Table 6.5 shows the percentage influence of the frequency bound on the runtime without this bound, and on the number of visited itemsets. Clearly, these two statistics are highly correlated. On the larger datasets, the negative impact on pruning is high, and hence the frequency bound results in a strong increase in runtime. On the other datasets, the impact of the frequency bound on pruning is limited, resulting in improved or unaffected runtimes. The results indicate that the maximal frequency bound is a useful optimization on smaller datasets, especially for smaller τ-values, but should be disabled for larger datasets.

6.8 Conclusion

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

We have then specialized the general setting for our dynamic notion of data quality, by introducing so-called forbidden itemsets. In order to discover such low-lift forbidden itemsets, we have provided an efficient algorithm, and established properties of the lift measure to use for pruning. Our experiments show that the algorithm is efficient, and illustrate that forbidden itemsets capture inconsistencies with high precision, while providing a concise representation of dirtiness in data.

CHAPTER 7

Repairing with Forbidden Itemsets

7.1 Introduction

In the previous chapter, we have introduced a dynamic notion of data quality, stating that data is clean if a constraint discovery algorithm does not detect violation. We have also introduced a constraint formalism, called forbidden itemsets, for high-precision detection of violations.

In this chapter, we address the biggest challenge with respect to our dynamic notion of data quality, which is to repair data in such a way that the discovery algorithm no longer detects violations. In the context of forbidden itemsets, this means that we have to suggest modifications to the data, such that after these modifications, no new forbidden itemsets are found. Of course, the number of possible modifications to the data increases exponentially with the amount of dirtiness. It is therefore not feasible to rediscover forbidden itemsets for every possible modification, to verify that no forbidden itemsets exist. We first introduce a general method to solve this problem, by computing up front which itemsets are at risk of becoming dirty, which we call *almost forbidden itemsets*. As such, by ensuring that a repair algorithm's modifications do not impact the almost forbidden itemsets, we can guarantee that the cleaned data does not contain any forbidden itemsets.

We then return to the concrete case of low-lift τ-forbidden itemsets, and present an algorithm for discovering almost forbidden itemsets with respect to the lift measure. We also discuss how the efficiency of this method can be improved further by repairing the dirty parts of the data in suitably sized batches, instead of all at once. To obtain candidate modifications, we again take inspiration from census data imputation methods that assume the presence of enough clean data [59] and take suggested modifications from similar, clean objects. The availability of clean data is commonly assumed in

repairing methods, either as a large part of the input data or, for example, in the form of master data [47, 51]. Our repair method is flexible in how these similarities are computed.

More formally, the FBIMINER algorithm from the previous chapter discovers a set of τ-forbidden itemsets that describe inconsistencies in $\mathcal{D}$. When this set is non-empty, $\mathcal{D}$ is regarded as dirty. In this chapter, we discuss how to clean $\mathcal{D}$. Following the general framework outlined in Section 6.2 we wish to compute a repair $\mathcal{D}'$ of $\mathcal{D}$ such that $\mathcal{D}'$ is clean, i.e., no τ-forbidden itemsets should be found in $\mathcal{D}'$. Due to the dynamic notion of data quality, this becomes much more challenging than in a traditional repair setting. To obtain repairs, we impute values from clean objects that differ minimally from a dirty object, in line with common practice in data imputation. This heuristic repair method aims to limit the difference between $\mathcal{D}'$ and $\mathcal{D}$.

7.1.1 Summary of Contributions

1. We introduce the concept of almost forbidden itemsets, allowing a repair algorithm to compute up front enough information to verify the validity of repair modifications. (Section 7.2)
2. We formalize a relaxed form of the lift measure, called the minimal possible lift after k modifications. This measure allows us to compute almost forbidden itemsets in our context of low-lift τ-forbidden itemsets. (Section 7.3)
3. We provide an efficient itemset mining algorithm for discovering almost forbidden itemsets using the minimal possible lift measure. The efficiency of this algorithm is again based on pruning properties, which we derive from the minimal possible lift measure. We further improve the efficiency by repairing dirty objects in batches of a suitable size, and derive guidelines for these sizes from our pruning properties. (Sections 7.4–7.6)
4. We introduce a flexible repair algorithm, which modifies dirty objects by taking cues from clusters of similar but clean objects. The correctness of the repair algorithm is formally proven, i.e., it is guaranteed that the result of a repair no longer contains any forbidden itemsets. (Section 7.7)
5. We experimentally evaluate our method on various real-world datasets, and show that our repair algorithm brings dirty data closer to ground truth. We additionally show that almost forbidden itemsets are necessary for an efficient repair algorithm, and that the suitable choice of batch size is crucial. Finally, we show that repairing in parallel can provide additional improvements in efficiency. (Section 7.8)

7.2 Avoiding New Unlikely Value Combinations

In this section we again temporarily consider a general likeliness function (see Section 6.3), since the ideas developed here may be of interest to like-

liness functions other than the lift measure. Given a dataset $\mathcal{D}$, a likeliness function L and a maximum likeliness threshold τ, let us denote the set of (L,τ)-forbidden itemsets by $\mathsf{FBI}(\mathcal{D},L,\tau)$. Let the *dirty* objects in $\mathcal{D}$, denoted by $\mathcal{D}_{dirty}$, be those objects in $\mathcal{D}$ that appear in the cover of an itemset in $\mathsf{FBI}(\mathcal{D},L,\tau)$. In other words, $\mathcal{D}_{dirty}$ consists of all objects that support an (L,τ)-forbidden itemset. The remaining set of clean objects in $\mathcal{D}$ is denoted by $\mathcal{D}_r$. The main goal of a repair algorithm is then to produce a dataset $\mathcal{D}'$ such that $\mathsf{FBI}(\mathcal{D}',L,\tau)$ is empty, i.e., such that $\mathcal{D}'$ is clean. We will consider datasets $\mathcal{D}'$ obtained from $\mathcal{D}$ by modifying all objects in $\mathcal{D}_{dirty}$. We first show by example that this is not a trivial problem.

Example 13. People typically graduate High School in the year that they turn 18. Depending on the timing of a census, there may be graduates who are still only 17 years old. In the Adult Census dataset, the itemset (AGE=<18, EDUCATION=HS-GRAD) is rare, with a lift ≈ 0.072. Assume that τ-forbidden itemsets were mined with $\tau = 0.07$. The itemset (AGE=<18, EDUCATION=HS-GRAD) is thus not considered forbidden, and rightly so. However, an object containing (AGE=<18, EDUCATION=HS-GRAD) could, for example, contain the forbidden itemset (AGE=<18, MARITALSTATUS=DIVORCED), where "MaritalStatus" is in error. If the repair algorithm were to change "Age" instead, the lift of (AGE=<18, EDUCATION=HS-GRAD) could drop to ≈ 0.068! This itemset will then become τ-forbidden in $\mathcal{D}'$, yielding again a dirty dataset. This illustrates that one has to be careful which modifications are carried out during repairing. $\diamondsuit$

A naive approach for avoiding new inconsistencies would be to compute $\mathsf{FBI}(\mathcal{D}',L,\tau)$ for each candidate repair $\mathcal{D}'$ of $\mathcal{D}$, and reject $\mathcal{D}'$ in the case that $\mathsf{FBI}(\mathcal{D}',L,\tau)$ is non-empty. In view of the possibly exponential number of candidate repairs, this approach is not feasible for all but the smallest datasets.

As an alternative, we propose to compute up front enough information to ensure cleanliness of multiple repairs. In particular, consider repairs $\mathcal{D}'$ obtained from $\mathcal{D}$ by only modifying the dirty objects in $\mathcal{D}_{dirty}$. We consider a modification of an object to refer to one or more changes in the itemset-representation of said object, similar to a set of modifications with the same tid in Chapter 5. This implies that any repair $\mathcal{D}'$ can be obtained by at most $|\mathcal{D}_{dirty}|$ modifications. More generally, assume that $\mathcal{D}'$ is obtained from $\mathcal{D}$ by modifying at most k dirty objects. Furthermore, assume that we can compute a lower bound $L_k(I,\mathcal{D})$ for $L(I,\mathcal{D}')$, the likeliness of the itemset I after repairing, *using only* k, $\mathcal{D}$, *and* I *itself*. In other words, it should hold that $L_k(I,\mathcal{D}) \leqslant L(I,\mathcal{D}')$ for *any* $\mathcal{D}'$ obtained from $\mathcal{D}$ by modifying at most k dirty objects. We can then define a set $\mathbb{A}$ of so-called *almost* (L,τ)*-forbidden itemsets*, i.e., itemsets that could become (L,τ)-forbidden after applying at most k modifications to $\mathcal{D}$.

Definition 27. Let $\mathcal{D}$ be a dataset, L a likeliness function, I an itemset and τ a maximum likeliness threshold. Let $L_k(I,\mathcal{D})$ denote a lower bound on $L(I,\mathcal{D}')$ for any $\mathcal{D}'$ obtained from $\mathcal{D}$ by at most k modifications, i.e., by replacing items in at most k dirty objects. Then, I is called a k*-almost* (L,τ)*-forbidden itemset* if $L_k(I,\mathcal{D}) \leqslant \tau$. $\square$

We emphasize that the function L_k should only depend on k, $\mathcal{D}$ and I, and not on the repair $\mathcal{D}'$ under consideration. Let $\mathbb{A}_\tau$ denote the set of all k-almost (L,τ)-forbidden itemsets. More formally,

$$\mathbb{A}_\tau := \{I \mid I \text{ is an itemset in } \mathcal{D} \text{ and } L_k(I,\mathcal{D}) \leqslant \tau\}.$$

Clearly, for any $\mathcal{D}'$ obtained from $\mathcal{D}$ by at most k modifications to dirty objects in $\mathcal{D}$, if $I \in \mathsf{FBI}(\mathcal{D}',L,\tau)$, then we have that $L(I,\mathcal{D}') \leqslant \tau$, and hence $L_k(I,\mathcal{D}) \leqslant \tau$. This leads to the following observation.

Proposition 12. Let $\mathcal{D}$ be a dataset and τ a maximum likeliness threshold. For any dataset $\mathcal{D}'$ obtained from $\mathcal{D}$ by modifying at most k dirty objects in $\mathcal{D}$, it holds that:

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

This proposition implies that if $\mathbb{A}_\tau$ can be computed efficiently, then we do not need to consider all possible repairs $\mathcal{D}'$ of $\mathcal{D}$, obtained by at most k modifications, in order to detect (L,τ)-forbidden itemsets in $\mathcal{D}'$. It eliminates the need for rediscovering itemsets for all possible repairs, one by one. We next explain how to compute L_k and almost forbidden itemsets when the likeliness function L is the lift measure.

7.3 Almost Forbidden Itemsets

In order to discover Almost Forbidden Itemsets when $L(I,\mathcal{D})$ is the lift measure, we need to answer the following question: *Given an itemset I in $\mathcal{D}$ and its* $\mathsf{lift}(I,\mathcal{D})$*, can I become τ-forbidden after k modifications to $\mathcal{D}$ have been made*? We answer this question by identifying a lower bound on the lift of an itemset I after k modifications to $\mathcal{D}$. This lower bound will be referred to as the *minimal possible lift of I after k modifications* and will be denoted by $\mathsf{mplift}_k(I,J,\mathcal{D})$ for some subset $J \subset I$, as will be explained below.

The crucial property of $\mathsf{mplift}_k(I,J,\mathcal{D})$ is that whenever $\mathsf{lift}(I,\mathcal{D}') \leqslant \tau$ for some $\mathcal{D}'$ obtained from $\mathcal{D}$ using at most k modifications, then $\mathsf{mplift}_k(I,J,\mathcal{D}) \leqslant \tau$ as well. In other words, the set of itemsets I such that $\mathsf{mplift}_k(I,J,\mathcal{D}) \leqslant \tau$ consists of all τ-forbidden itemsets in any $\mathcal{D}'$, and possibly more.

To start with, we consider $k = 1$. That is, we want to lower bound the lift of itemsets in any $\mathcal{D}'$ obtained from $\mathcal{D}$ after one modification. Clearly, for all subsets $J \subset I$, we have:

$$\mathsf{lift}(I,\mathcal{D}') \geqslant \frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D}')}{\mathsf{supp}(J,\mathcal{D}') \times \mathsf{supp}(I \setminus J,\mathcal{D}')}.$$

We first consider the case when $\mathsf{supp}(I,\mathcal{D}) > 0$. Recall that our goal is to prevent the existence of τ-forbidden itemsets in $\mathcal{D}'$. Thus, we are concerned with lower bounding $\mathsf{lift}(I,\mathcal{D}')$ in those $\mathcal{D}'$ that contain I, i.e., $\mathsf{supp}(I,\mathcal{D}') > 0$.

Consider $\mathcal{D}'$ obtained from $\mathcal{D}$ by one modification. Such a single modification can only have as effect that either $\mathsf{supp}(I,\mathcal{D}') = \mathsf{supp}(I,\mathcal{D})$, $\mathsf{supp}(I,\mathcal{D}') =$

$\mathsf{supp}(I,\mathcal{D})+1$ or $\mathsf{supp}(I,\mathcal{D}') = \mathsf{supp}(I,\mathcal{D})-1$. The same changes are possible for $\mathsf{supp}(J,\mathcal{D}')$ and $\mathsf{supp}(I \setminus J,\mathcal{D}')$. For convenience, let us denote each of these possible changes to the three itemsets I, J and $I \setminus J$ by a triple $(\Delta_I,\Delta_J,\Delta_{I\setminus J})$ such that each of the components can either be 0 (no change), $+1$ (an increase with one) or -1 (a decrease with one). We thus have that for every possible $\mathcal{D}'$, we can identify $(\Delta_I,\Delta_J,\Delta_{I\setminus J})$ such that

$$\mathsf{lift}(I,\mathcal{D}') \geqslant \frac{|\mathcal{D}| \times (\mathsf{supp}(I,\mathcal{D})+\Delta_I)}{(\mathsf{supp}(J,\mathcal{D})+\Delta_J) \times (\mathsf{supp}(I \setminus J,\mathcal{D})+\Delta_{I\setminus J})}.$$

Note that $|\mathcal{D}'| = |\mathcal{D}|$, since we do not consider insertions or deletions. The previous inequality holds, of course, only when neither $\mathsf{supp}(J,\mathcal{D})+\Delta_J$ nor $\mathsf{supp}(I \setminus J,\mathcal{D})+\Delta_{I\setminus J}$ is zero, to rule out division by zero. Additionally, in order to use the lower bound for pruning, we need to obtain a positive, non-zero lower bound. We achieve this by excluding cases where $\mathsf{supp}(I,\mathcal{D})+\Delta_I$ is zero, e.g., when $\mathsf{supp}(I,\mathcal{D}) = 1$ and $\Delta_I = -1$. Indeed, this would mean that I does not have a support in $\mathcal{D}'$ and hence cannot be forbidden. The other cases that may lead to division by zero or negative values concern cases when $\mathsf{supp}(I,\mathcal{D})$, $\mathsf{supp}(J,\mathcal{D})$ or $\mathsf{supp}(I \setminus J,\mathcal{D})$ is 0 and Δ_I, Δ_J or $\Delta_{I\setminus J}$ is either 0 or -1. We observe, however, that we assumed that $\mathsf{supp}(I,\mathcal{D}) > 0$. Hence, $\mathsf{supp}(J,\mathcal{D}) > 0$ and $\mathsf{supp}(I \setminus J,\mathcal{D}) > 0$ for any $J \subset I$. So, the nominator and the denominator will never take a value $\leqslant 0$, guaranteeing a positive, non-zero lower bound.

To get a lower bound for $\mathsf{lift}(I,\mathcal{D}')$ that is independent of $\mathcal{D}'$ (or equivalently the choice of $(\Delta_I,\Delta_J,\Delta_{I\setminus J})$) we simply consider all possibilities and take the minimum value:

$$\min_{(\Delta_I,\Delta_J,\Delta_{I\setminus J})} \frac{|\mathcal{D}| \times (\mathsf{supp}(I,\mathcal{D})+\Delta_I)}{(\mathsf{supp}(J,\mathcal{D})+\Delta_J) \times (\mathsf{supp}(I \setminus J,\mathcal{D})+\Delta_{I\setminus J})}. \qquad (\ddagger)$$

As above, triples $(\Delta_I,\Delta_J,\Delta_{I\setminus J})$ that make either $\mathsf{supp}(I,\mathcal{D})+\Delta_I$, $\mathsf{supp}(J,\mathcal{D})+\Delta_J$ or $\mathsf{supp}(I \setminus J,\mathcal{D})+\Delta_{I\setminus J}$ zero are excluded in the computation of (‡). The crucial property of (‡) is that it is lower than $\mathsf{lift}(I,\mathcal{D}')$ for any $\mathcal{D}'$ obtained from $\mathcal{D}$ after one modification, for all itemsets I with $\mathsf{supp}(I,\mathcal{D}') > 0$.

To define the notion of minimal possible lift, we carry out some case analysis. To simplify notation, we let $\sigma_I = \mathsf{supp}(I,\mathcal{D})$, $\sigma_J = \mathsf{supp}(J,\mathcal{D})$ and $\sigma_{I\setminus J} = \mathsf{supp}(I \setminus J,\mathcal{D})$. Assuming that $(\Delta_I,\Delta_J,\Delta_{I\setminus J})$ minimizes (‡) and letting $\sigma^1_I = \sigma_I+\Delta_I$, $\sigma^1_J = \sigma_J+\Delta_J$ and $\sigma^1_{I\setminus J} = \mathsf{supp}(I \setminus J,\mathcal{D})+\Delta_{I\setminus J}$, the expression (‡) can be written as

$$\frac{|\mathcal{D}| \times \sigma^1_I}{\sigma^1_J \times \sigma^1_{I\setminus J}}.$$

Up to this point, we have assumed that $\mathsf{supp}(I,\mathcal{D}) > 0$. When $\mathsf{supp}(I,\mathcal{D}) = 0$, and since we are only interested to lower bound the lift of I in $\mathcal{D}'$ when $\mathsf{supp}(I,\mathcal{D}') > 0$, this implies that we may assume that $\mathsf{supp}(I,\mathcal{D}') = 1$, and thus $\Delta_I = +1$. Furthermore, this also implies that $\mathsf{supp}(J,\mathcal{D}') \geqslant \mathsf{supp}(J,\mathcal{D})$ for any $J \subset I$, and $\mathsf{supp}(J,\mathcal{D}') = \mathsf{supp}(J,\mathcal{D})+1$ for at least one $J \subset I$. We thus have three possible triples to describe these changes: $(+1,0,+1)$, $(+1,+1,0)$ and $(+1,+1,+1)$. Clearly, the latter causes the greatest increase in the denominator, leading to the minimal lift. We thus have that $\mathsf{lift}(I,\mathcal{D}')$ is greater than or

equal to

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

and there is no need to lower bound $\mathsf{lift}(I,\mathcal{D}')$ since its exact value can be computed based on the supports of I, J and $I \setminus J$ in $\mathcal{D}$ alone. Using the notation introduced earlier, and denoting $\sigma_I^1 = \sigma_I + 1$, $\sigma_J^1 = \sigma_J + 1$ and $\sigma_{I\setminus J}^1 = \sigma_{I\setminus J} + 1$, we have that:

$$\mathsf{lift}(I,\mathcal{D}') \geqslant \frac{|\mathcal{D}| \times \sigma_I^1}{\sigma_J^1 \times \sigma_{I\setminus J}^1}.$$

Suppose next that $k > 1$. We have just identified, for $k = 1$, those updates to σ_I, σ_J and $\sigma_{I\setminus J}$ that lead to the worst possible decrease in lift of I when considering all possible $\mathcal{D}'$ obtained from $\mathcal{D}$ after one modification; i.e., σ_I^1, σ_J^1 and $\sigma_{I\setminus J}^1$, respectively. Intuitively, σ_I^1, σ_J^1 and $\sigma_{I\setminus J}^1$ correspond to supports of itemsets I, J and $I \setminus J$ in some $\mathcal{D}_1'$ obtained from $\mathcal{D}$ after one modification. We can now repeat the analysis for the second modification, starting from σ_I^1, σ_J^1, $\sigma_{I\setminus J}^1$ and $\mathcal{D}_1'$, instead of σ_I, σ_J and $\sigma_{I\setminus J}$. This results again in updated supports σ_I^2, σ_J^2 and $\sigma_{I\setminus J}^2$ that lead to the worst possible decrease in lift of I in $\mathcal{D}_1'$ after one modification, or equivalently, of the lift of I in $\mathcal{D}$ after two modifications. Continuing in this way, we can recursively compute σ_I^k, σ_J^k and $\sigma_{I\setminus J}^k$ such that for any $\mathcal{D}'$ obtained from $\mathcal{D}$ by at most k modifications:

$$\mathsf{lift}(I,\mathcal{D}') \geqslant \frac{|\mathcal{D}| \times \sigma_I^k}{\sigma_J^k \times \sigma_{I\setminus J}^k}.$$

The previous discussion leads to our definition of $\mathsf{mplift}_k(I,J,\mathcal{D})$, the minimal possible lift of I after k modifications. The definition formalizes the analysis carried out earlier, and additionally includes a substantial simplification. Indeed, as we will formally show below, we only need to consider a limited number (at most two) possible triples $(\Delta_I, \Delta_J, \Delta_{I\setminus J})$ in order to compute (‡). In particular, it suffices to consider $(-1,0,-1)$ and $(0,+1,0)$, as one of these always leads to the largest reduction in lift. We start by showing that these two triples suffice to compute expression (‡).

Proposition 13. Let $\mathcal{D}$ be a dataset, and I and J itemsets such that $\mathsf{supp}(J,\mathcal{D}) \leqslant \mathsf{supp}(I \setminus J,\mathcal{D})$. It then follows that:

▷ If $\mathsf{supp}(I,\mathcal{D}) > 1$ and $\mathsf{supp}(I \setminus J,\mathcal{D}) > 1$, then

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

▷ Otherwise, if $\mathsf{supp}(I,\mathcal{D}) = 1$, or $\mathsf{supp}(I \setminus J,\mathcal{D}) = 1$, then

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

$$f(x,y,z) = \begin{cases} g(x,y,z) & y \leqslant z \\ g(x,z,y) & y > z. \end{cases}$$

$$g(x,y,z) = \begin{cases} (x-1,y,z-1) & \text{if } \frac{x-1}{y\times(z-1)} \leqslant \frac{x}{(y+1)\times z} \text{ and } x-1>0, z-1>0; \\ (x,y+1,z) & \text{if } \frac{x-1}{y\times(z-1)} > \frac{x}{(y+1)\times z} \text{ and } x-1>0, z-1>0; \\ & \textbf{or if } x=1 \text{ or } z=1; \\ (x+1,y+1,z+1) & \text{if } x=0. \end{cases}$$

Figure 7.1: Definition of the recursive function f that identifies those updates to x, y and z that will lead to the largest reduction in the lift.

Proof. The idea behind the proof is to show that for all $(\Delta_I, \Delta_J, \Delta_{I\setminus J})$,

$$\frac{|\mathcal{D}| \times (\mathsf{supp}(I,\mathcal{D}) + \Delta_I)}{(\mathsf{supp}(J,\mathcal{D}) + \Delta_J) \times (\mathsf{supp}(I\setminus J,\mathcal{D}) + \Delta_{I\setminus J})}$$

is either larger than

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

which corresponds to the triple $(-1,0,-1)$, or larger than

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

which corresponds to the triple $(0,+1,0)$.

These inequalities can be verified by a simple, yet tedious, case analysis. We defer the details to Appendix C.1. □

Next, we formalize the recursive selection of the updates σ_I^i, σ_J^i, and $\sigma_{I\setminus J}^i$ of σ_I, σ_J and $\sigma_{I\setminus J}$, respectively, for $i = 1,2,\ldots,k$. To this aim we define a function $f(x,y,z)$ from triples in $\mathbb{N}^3$ to triples in $\mathbb{N}^3$, as shown in Figure 7.1. Here x, y and z correspond to σ_I, σ_J and $\sigma_{I\setminus J}$, respectively.

First, $f(x,y,z)$ is defined in terms of another function $g(x,y,z)$ which just ensures that the second argument is smaller or equal than the third argument. This is needed to reduce the computation of expression (‡) to the two triples mentioned earlier. Indeed, observe that Proposition 13 requires $\mathsf{supp}(J,\mathcal{D}) \leqslant \mathsf{supp}(I\setminus J,\mathcal{D})$ and hence we either call $g(x,y,z)$ if $y \leqslant z$, or call $g(x,z,y)$ if $y > z$. The latter simply corresponds to changing the role of J and $I\setminus J$ in the analysis.

Then, $g(x,y,z)$ captures the case analyses explained previously. For example, when $x = 0$ ($\sigma_I = 0$) we need to update x, y and z according to expression (††). That is, $g(x,y,z) = (x+1,y+1,z+1)$. Similarly, when either $x = 1$ or $z = 1$, letting $g(x,y,z) = (x,y+1,z)$ results in the minimization of (‡)

as described in Proposition 13. In all other cases, $g(x,y,z) = (x-1, y, z-1)$ or $g(x,y,z) = (x, y+1, z)$ depending on which of the two updates to the supports minimizes (‡) as described in Proposition 13. The former case corresponds to the triple $(-1,0,-1)$, the latter to triple $(0,+1,0)$. It should be clear that $f(x,y,z)$ correctly updates the values x, y and z. That is, it returns updated values that compute (‡) or (††) when one modification is considered. By recursively applying the function f we ensure that $(x^i, y^i, z^i) = f(x^{i-1}, y^{i-1}, z^{i-1})$ correspond to updates that maximally reduce the lift after i modifications.

Given this, we next define the notion of minimal possible lift of an itemset after k modifications.

Definition 28. Let $\mathcal{D}$ be a dataset. Let I be an itemset and let $J \subset I$. Denote by $\sigma_I^0 = \mathsf{supp}(I,\mathcal{D})$, $\sigma_J^0 = \mathsf{supp}(J,\mathcal{D})$ and $\sigma_{I\setminus J}^0 = \mathsf{supp}(I \setminus J, \mathcal{D})$. Let $f(x,y,z)$ be the function from $\mathbb{N}^3 \to \mathbb{N}^3$ as defined in Figure 7.1. Let $(\sigma_I^i, \sigma_J^i, \sigma_{I\setminus J}^i) = f(\sigma_I^{i-1}, \sigma_J^{i-1}, \sigma_{I\setminus J}^{i-1})$, for $i = 1, 2, \ldots, k$. Then the *minimal possible lift of I after k modifications* is given by

$$\mathsf{mplift}_k(I,J,\mathcal{D}) := \frac{|\mathcal{D}| \times \sigma_I^k}{\sigma_J^k \times \sigma_{I\setminus J}^k}. \qquad \square$$

Since (‡) and (††) correspond to lower bounds of the lifts, Proposition 13 and the construction of the function $f(x,y,z)$ guarantee that, for any $\mathcal{D}'$ obtained from $\mathcal{D}$ after at most k modifications:

$$\mathsf{mplift}_k(I,J,\mathcal{D}) \leqslant \mathsf{lift}(I,\mathcal{D}').$$

We observe that this holds for any subset $J \subset I$. In the rest of this chapter, we always assume that J is chosen such that $\mathsf{lift}(I,\mathcal{D}) = \frac{|\mathcal{D}| \times \mathsf{supp}(I,\mathcal{D})}{\mathsf{supp}(J,\mathcal{D}) \times \mathsf{supp}(I\setminus J,\mathcal{D})}$ when $\mathsf{supp}(I,\mathcal{D}) > 0$. Intuitively, picking the subset J that currently maximizes the lift of I is likely to also maximize the minimal possible lift of I after modifications, i.e., to lead to the tightest possible bound on the lift of I after k modifications. If $\mathsf{supp}(I,\mathcal{D}) = 0$, we may still identify the subset J that minimizes $\mathsf{supp}(J,\mathcal{D}) \times \mathsf{supp}(I \setminus J, \mathcal{D})$, if a J exists with $\mathsf{supp}(J,\mathcal{D}) > 0$ and $\mathsf{supp}(I \setminus J, \mathcal{D}) > 0$. Otherwise, we use the trivial lower bound $\mathsf{supp}(J,\mathcal{D}) = \mathsf{supp}(I \setminus J, \mathcal{D}) = 1$. Linking back to the previous section, if we consider

$$\mathbb{A}_\tau := \{I \mid I \text{ is an itemset in } \mathcal{D} \text{ and } \mathsf{mplift}_k(I,J,\mathcal{D}) \leqslant \tau\},$$

then $\mathsf{mplift}_k(I,J,\mathcal{D})$ satisfies the conditions that resulted in Proposition 12. Hence, we have that $\mathsf{FBI}(\mathcal{D}',\tau) \subseteq \mathbb{A}_\tau$, as desired.

7.4 Repair-oblivious Pruning Properties

As explained at the beginning of this chapter, we will develop an algorithm, called A-FBIMINER, for discovering $\mathbb{A}_\tau$ (or more precisely a subset of $\mathbb{A}_\tau$ as will be explained shortly). This algorithm is similar in spirit to FBIMINER, using mplift rather than lift. However, mplift is too lenient as the basis for

an efficient discovery algorithm, as it is based on worst-case scenarios. After all, it accommodates for all possible repairs obtained by at most k modifications. As a consequence, $\mathbb{A}_\tau$ may contain too many itemsets. Since the A-FBIMINER algorithm also deploys a depth-first search strategy, we next explore how $\mathbb{A}_\tau$ can be reduced by pruning itemsets that cannot be τ-forbidden in any repair $\mathcal{D}'$. We will design the A-FBIMINER algorithm based on such repair-oblivious pruning properties and hence, A-FBIMINER will return a subset $\mathbb{A}$ of $\mathbb{A}_\tau$, yet without violating property (†). That is, it still holds that $\mathsf{FBI}(\mathcal{D}', \tau) \subseteq \mathbb{A}$ for any $\mathcal{D}'$.

Recall that algorithm A-FBIMINER is to mine almost forbidden itemsets without looking at specific modifications, i.e., only $\mathcal{D}$ and an upper bound k on the number of modified objects is available. Clearly k is at most $|\mathcal{D}_{dirty}|$. Additionally, the set $\mathcal{D}_{dirty}$ is known up front, and may be used to refine the mplift measure. We next adapt the pruning strategies from FBIMINER, by revising the underlying properties to take possible modifications into account. Since clean objects are never modified, a tight bound on the support of an itemset in any $\mathcal{D}'$, obtained from $\mathcal{D}$ by at most k modifications, can be obtained. Indeed, we observe that for any itemset I, the following holds:

$$\mathsf{supp}(I, \mathcal{D}_r) \leqslant \mathsf{supp}(I, \mathcal{D}') \leqslant \mathsf{supp}(I, \mathcal{D}_r) + k. \tag{§}$$

An immediate consequence is that A-FBIMINER must also consider itemsets I with $\mathsf{supp}(I, \mathcal{D}_r) = 0$ as these can become supported in repairs. Furthermore, we can now modify Propositions 7 and 8. In the following, $\mathcal{D}'$ denotes a dataset obtained from $\mathcal{D}$ by at most k modifications to objects in $\mathcal{D}_{dirty}$.

Proposition 14. For any two itemsets I and J such that $I \subset J$, if J is a τ-forbidden itemset in $\mathcal{D}'$, then we have that $\mathsf{supp}(I, \mathcal{D}_r) \geqslant \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau} - k$.

Proof. We show this by contradiction. Let J be a τ-forbidden itemset in $\mathcal{D}'$ for $\tau < 1$ and assume for the sake of contradiction that there exists a $I \subset J$ with $\mathsf{supp}(I, \mathcal{D}_r) < \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau} - k$. From the inequalities (§) it follows that:

$$\mathsf{supp}(I, \mathcal{D}') \leqslant \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau} - k + k$$

And hence $\mathsf{supp}(I, \mathcal{D}') \leqslant \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau}$. According to Proposition 7, this implies that J cannot be a τ-forbidden itemset in $\mathcal{D}'$, which contradicts our initial assumption. Hence, every subset I of a τ-forbidden itemset J in $\mathcal{D}'$ must satisfy $\mathsf{supp}(I, \mathcal{D}_r) \geqslant \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}')}{\sigma^{max}_{I,\mathcal{D}'} \times \tau} - k$. □

Proposition 15. For any three itemsets I, J and K such that $I \subset J \subseteq K$, if K is a τ-forbidden itemset in $\mathcal{D}'$, then $\mathsf{supp}(I, \mathcal{D}_r) - \mathsf{supp}(J, \mathcal{D}_r) \geqslant \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} - k$.

Proof. We show this by contradiction. Let K be a τ-forbidden itemset in $\mathcal{D}'$ for $\tau < 1$ and assume for the sake of contradiction that there exist subsets

$I \subset J \subseteq K$ with $\mathsf{supp}(I,\mathcal{D}_r) - \mathsf{supp}(J,\mathcal{D}_r) < \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} - k$. From the inequalities (§) it follows that

$$k \leqslant \mathsf{supp}(I,\mathcal{D}') - \mathsf{supp}(I,\mathcal{D}_r)$$

By substituting this inequality for k, we obtain:

$$\cancel{\mathsf{supp}(I,\mathcal{D}_r)} - \mathsf{supp}(J,\mathcal{D}_r) < \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} - \mathsf{supp}(I,\mathcal{D}') + \cancel{\mathsf{supp}(I,\mathcal{D}_r)}$$

$$\mathsf{supp}(I,\mathcal{D}') - \mathsf{supp}(J,\mathcal{D}_r) < \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|}$$

Since $\mathsf{supp}(J,\mathcal{D}_r) \leqslant \mathsf{supp}(J,\mathcal{D}')$, it follows that

$$\mathsf{supp}(I,\mathcal{D}') - \mathsf{supp}(J,\mathcal{D}_r) \leqslant \mathsf{supp}(I,\mathcal{D}') - \mathsf{supp}(J,\mathcal{D}')$$

Hence, we may conclude that

$$\mathsf{supp}(I,\mathcal{D}') - \mathsf{supp}(J,\mathcal{D}') < \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|}$$

This violates Proposition 8, implying that no superset of J can be τ-forbidden in $\mathcal{D}'$. This contradicts our assumption that $K \supseteq J$ is τ-forbidden in $\mathcal{D}'$. □

In order to use these propositions without relying on knowledge about supports in $\mathcal{D}'$ (recall that we do not know which $\mathcal{D}'$ we are considering), we need to replace $\mathsf{supp}(J,\mathcal{D}')$ and $\sigma^{max}_{I,\mathcal{D}'}$ by quantities derived from $\mathcal{D}$ alone.

For Proposition 14, similarly to the pruning strategies for FBIMiner, we again use the trivial lower bound $\mathsf{supp}(J,\mathcal{D}') \geqslant 1$. Also, note that $\sigma^{max}_{I,\mathcal{D}_r}$ can be computed, and it holds that $\sigma^{max}_{I,\mathcal{D}'} \leqslant \sigma^{max}_{I,\mathcal{D}_r} + k$. Hence, Proposition 14 is used to prune supersets of I whenever

$$\mathsf{supp}(I,\mathcal{D}_r) < \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r} + k) \times \tau} - k. \tag{P1}$$

Similarly, Proposition 15 prunes itemsets J if there is a subset I of J such that

$$\mathsf{supp}(I,\mathcal{D}_r) - \mathsf{supp}(J,\mathcal{D}_r) < \frac{1}{\tau} - \frac{\sigma^{max}_{I,\mathcal{D}'}}{|\mathcal{D}|} - k.$$

Since it holds that $\sigma^{max}_{I,\mathcal{D}'} \leqslant |\mathcal{D}|$, we may avoid the approximation of $\sigma^{max}_{I,\mathcal{D}'}$ altogether, and instead simplify the expression to:

$$\mathsf{supp}(I,\mathcal{D}_r) - \mathsf{supp}(J,\mathcal{D}_r) < \frac{1}{\tau} - 1 - k. \tag{P2}$$

Proposition 9 also needs to be modified to account for possible modifications. Recall that this proposition was based on the anti-monotonicity of the lift denominator. In order to preserve this property under modifications, we need to compute the worst case increase in the denominator of the lift measure. Since this denominator consists of a product of supports, we apply upper bounds on both supports:

Proposition 16. For any two itemsets I and J such that $I \subset J$, it holds that

$$\mathsf{lift}(J,\mathcal{D}') \geqslant \frac{\mathsf{supp}(J,\mathcal{D}') \times |\mathcal{D}|}{\min\limits_{S\subset I}\big\{(\mathsf{supp}(S,\mathcal{D}_r)+k)\times(\mathsf{supp}(I\setminus S,\mathcal{D}_r)+k)\big\}}.$$

Proof. This is a straightforward extension of Proposition 9. Indeed, observe that

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

By applying the inequalities $\mathsf{supp}(S,\mathcal{D}') \leqslant \mathsf{supp}(S,\mathcal{D}_r)+k$ and $\mathsf{supp}(I\setminus S,\mathcal{D}') \leqslant \mathsf{supp}(I\setminus S,\mathcal{D}_r)+k$, if follows that $\mathsf{lift}(J,\mathcal{D}') \geqslant$

$$\frac{\mathsf{supp}(J,\mathcal{D}') \times |\mathcal{D}|}{\min\limits_{S\subset I}\big\{(\mathsf{supp}(S,\mathcal{D}_r)+k)\times(\mathsf{supp}(I\setminus S,\mathcal{D}_r)+k)\big\}}.$$

□

To make use of this proposition for pruning, without knowing $\mathcal{D}'$, we set $\mathsf{supp}(J,\mathcal{D}')=1$ and hence prune I's supersets whenever

$$\frac{|\mathcal{D}|}{\min\limits_{S\subset I}\big\{(\mathsf{supp}(S,\mathcal{D}_r)+k)\times(\mathsf{supp}(I\setminus S,\mathcal{D}_r)+k)\big\}} \geqslant \tau \tag{P3}$$

Finally, we update Proposition 10 to obtain the maximal support in $\mathcal{D}_r$ of a τ-forbidden itemset in any $\mathcal{D}'$ obtained by modifying $\mathcal{D}$. Recall that this is not really a pruning strategy, but provides a way of avoiding certain unnecessary lift computations.

Proposition 17. For any itemset I that is τ-forbidden in $\mathcal{D}'$, it holds that $\mathsf{supp}(I,\mathcal{D}_r) \leqslant |\mathcal{D}| \times (\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2}-\frac{1}{\tau}} - 1)$.

Proof. Proposition 10 states that, for I to be τ-forbidden in $\mathcal{D}'$, it must hold that:

$$\mathsf{freq}(I,\mathcal{D}') \leqslant \frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2}-\frac{1}{\tau}} - 1$$

In terms of support, this implies that

$$\mathsf{supp}(I,\mathcal{D}') \leqslant |\mathcal{D}| \times (\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2}-\frac{1}{\tau}} - 1)$$

From the inequalities in (§), we use the fact that $\mathsf{supp}(I,\mathcal{D}') \geqslant \mathsf{supp}(I,\mathcal{D}_r)$ to obtain that:

$$\mathsf{supp}(I,\mathcal{D}_r) \leqslant |\mathcal{D}| \times (\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2}-\frac{1}{\tau}} - 1).$$ □

The three pruning strategies (P1), (P2) and (P3) allow substantial pruning when mining almost forbidden itemsets, similarly as their counterparts for FBIMINER. We will denote by $\mathbb{A}$ the set of almost forbidden itemsets returned by A-FBIMINER with these pruning strategies enabled. However, observe that k impacts the pruning power of (P1) and (P2). Indeed, when k is "chosen" such that the upper bounds in (P1) and (P2) do not impose any constraints on the supports of the itemsets involved, these pruning strategies are of no use. For example, suppose that k takes the maximal value, i.e., $k = |\mathcal{D}_{dirty}|$. Then, it is likely that (P1) and (P2) do not allow for any pruning. Worse still, the minimal possible lift, which also depends on k, will declare many itemsets to be almost forbidden, since the accuracy of this lower bound wanes as k increases. Although A-FBIMINER will need to be run only once to obtain the set $\mathbb{A}$ and all dirty objects can be repaired based on $\mathbb{A}$, this set will be big and inefficient to compute (due to lack of pruning). On the other hand, when $k = 1$, strong pruning will be possible and $\mathbb{A}$ will be of reasonable size. However, to clean all dirty objects, we need to deal with them one-at-a-time, re-running A-FBIMINER for $k = 1$ after a single dirty object is repaired. To attain maximal efficiency for algorithm A-FBIMINER we therefore propose, in the following section, to repair dirty objects in batches of limited size k.

7.5 Repairing In Batches

Instead of repairing all k objects at once, we now consider a partitioning of $\mathcal{D}_{dirty}$ into blocks of a size r. We want to optimize the trade-off between the runtime of A-FBIMINER and the number of runs of A-FBIMINER. The question, of course, is what block size to select. We already described block sizes $r = 1$ and $r = |\mathcal{D}_{dirty}|$, i.e., one-by-one and all-at-once. The question we next wish to answer is, how do the intermediate block sizes influence our pruning capabilities, and hence our runtime? We therefore analyze Proposition 14 and Proposition 15, replacing k with r, in order to identify the ranges for r in which pruning will still be possible.

We first turn our attention to Proposition 14. Let I be the itemset under consideration. Pruning strategy (P1) states that I's supersets can be pruned if $\mathsf{supp}(I,\mathcal{D}_r) < \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau} - r$. This is useful only if $\frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau} - r > 0$, since it trivially holds that $\mathsf{supp}(I,\mathcal{D}_r) \geqslant 0$. In order to perform any pruning with strategy (P1), we thus need to choose $r < \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau}$.

From Proposition 15, we derived pruning strategy (P2), stating that I's supersets may be pruned if $\mathsf{supp}(I,\mathcal{D}_r) - \mathsf{supp}(J,\mathcal{D}_r) < \frac{1}{\tau} - 1 - r$. Again, $\mathsf{supp}(I,\mathcal{D}_r) - \mathsf{supp}(J,\mathcal{D}_r) \geqslant 0$ holds trivially, and hence this strategy is only useful if $\frac{1}{\tau} - 1 - r > 0$. We thus require a block size $r < \frac{1}{\tau} - 1$ to perform any pruning using strategy (P2).

Moreover, it holds that $\frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)} \geqslant 1$, since $\sigma^{max}_{I,\mathcal{D}_r} \leqslant |\mathcal{D}_r|$, $|\mathcal{D}_r| + |\mathcal{D}_{\mathsf{dirty}}| = |\mathcal{D}|$, and $r \leqslant |\mathcal{D}_{\mathsf{dirty}}|$. It is then easy to see that $\frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau} > \frac{1}{\tau} - 1$. This implies that pruning with (P1) is always possible if pruning with (P2) is possible, but

not vice versa. To have *any* pruning with these strategies, a block size $r < \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau}$ is thus required.

The pruning strategy (P3) makes use of the number of dirty objects r in computing the values $(\mathsf{supp}(S,\mathcal{D}_r)+r)$ and $(\mathsf{supp}(I\setminus S,\mathcal{D}_r)+r)$, i.e., in bounding the support of I's subsets. Since the applicability of (P3) depends on the actual values of $\mathsf{supp}(S,\mathcal{D}_r)$ and $\mathsf{supp}(I\setminus S,\mathcal{D}_r)$, relative to each other, it is hard to gauge the impact of the block size r on this pruning strategy, and we cannot derive a maximal block size for which this strategy is useful. Of course, the general idea that a lower r improves pruning power holds for this strategy as well.

As a consequence, there is no universally optimal block size r, and it is difficult to decide up front. The specifics of the data and even the choice of which objects to include in a partition of r objects all impact the pruning power. It is also important to note that the block sizes derived above guarantee that the associated propositions are *applicable*, but they will still offer *reduced pruning power* in comparison to smaller block sizes. In the experimental section, we show that $r = \frac{1}{2\tau}$, half of the maximal size for which (P2) is applicable, provides a sensible default value of r. On the other hands, using $r = \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau} - 1$, the maximal size for any pruning with (P1), improves the runtime on datasets with a small number of attributes.

7.6 Mining Almost Forbidden Itemsets

The algorithm A-FBIMINER for mining almost τ-forbidden itemsets is identical in structure to FBIMINER, as shown in Algorithm 11. Itemsets are again discovered in a depth-first way, using a reverse pre-order traversal. The algorithm takes as input arguments a projected dataset in vertical layout $\mathcal{D}{\downarrow}[P]$, consisting of all objects which contain a prefix itemset P, and the lift threshold τ. Additionally, we pass the number k of dirty objects under consideration as a fourth argument. We next discuss the difference between the algorithms in terms of lift computation and pruning, and conclude this section with a proof of A-FBIMINER's correctness.

The most important difference between A-FBIMINER and FBIMINER is that, for every processed itemset I, we now compute the mplift measure instead of regular lift. Just like the lift measure, the mplift measure requires access to all subsets of I, which we store during the traversal using a prefix tree. The procedure for computing mplift is invoked on line 12. If the itemset I under consideration cannot be τ-forbidden in any $\mathcal{D}'$ according to Proposition 17, because it is too frequent in $\mathcal{D}_r$, we avoid computing its mplift on line 9.

The procedure for mplift first computes the actual lift of the itemset I, in order to identify a subset $S \subset I$ such that

$$\mathsf{lift}(I,\mathcal{D}) = |D| \times \mathsf{supp}(I,\mathcal{D}) / (\mathsf{supp}(S,\mathcal{D}) \times \mathsf{supp}(I\setminus S,\mathcal{D})).$$

In other words, we first find the partition that currently maximizes the lift, as explained earlier (just after Definition 28). Using the supports of I, S and $I\setminus S$,

the function f from Figure 7.1 in Section 7.3 computes the worst case configuration of these supports after k modifications. This configuration then gives us $\mathsf{mplift}_k(I)$. If mplift is lower than τ, then I is added to the set $\mathbb{A}$ consisting of almost FBIs (lines 12-13).

The pruning strategies employed by A-FBIMINER are equivalent to those in FBIMINER, except that the revised pruning strategies presented in Section 7.4 are used. First, Proposition 15 implies that non-generator itemsets can be pruned only when $k \leqslant \frac{1}{\tau} - 1$ (line 5). If this condition is satisfied, then procedures ISGENERATOR and STOREGENERATOR are used for pruning, exactly as before.

Similarly to the pruning strategies in FBIMINER, the pruning conditions for A-FBIMINER are used to prune all supersets of the current itemset I. On line 10, we detect whether a subset of I has previously been pruned. If this is the case, then I and all of its supersets may be pruned as well. If no subsets have been pruned, we are able to compute the mplift and lift measures of I. Using the denominators considered during lift computation, we can invoke pruning (line 15) based on the anti-monotonicity of the lift denominator as stated in Proposition 16. The minimum support in $\mathcal{D}_r$ of subsets of itemsets which are τ-forbidden in $\mathcal{D}'$ (Proposition 14) is verified on line 17, after processing I itself. If $\mathsf{supp}(I, \mathcal{D}_r)$ does not satisfy this condition, then all subsets of J are pruned from the search tree. Finally, extensions J of I that do not satisfy the minimum reduction in lift, based on Proposition 15, are discarded on line 25, when computing their supports, similar to FBIMINER.

The computation of supports in A-FBIMINER again makes use of set intersections of the covers of the items. These supports are computed in $\mathcal{D}$. Note, however, that A-FBIMINER also needs to consider itemsets I for which $\mathsf{supp}(I, \mathcal{D}) = 0$. Indeed, the check $\mathsf{supp}(J, \mathcal{D}) > 0$ on line 23 of FBIMINER has disappeared. For our pruning strategies, we also require the support of the itemsets in $\mathcal{D}_r$. We compute these by mining the supports of the itemsets in $\mathcal{D}_{dirty}$, since this set is typically much smaller than $\mathcal{D}_r$. Supports in $\mathcal{D}_{dirty}$ are also computed using set intersections of the covers of the items. The value $\mathsf{supp}(I, \mathcal{D}_r)$ is eventually obtained as $\mathsf{supp}(I, \mathcal{D}) - \mathsf{supp}(I, \mathcal{D}_{dirty})$ (line 24).

Proposition 18. Let $\mathcal{D}$ be a dataset and τ a maximum lift threshold. Algorithm A-FBIMINER$(\mathcal{D}\downarrow[\emptyset], \emptyset, k, \tau)$ returns a set $\mathbb{A}$ such that, if $\mathsf{lift}(J, \mathcal{D}') \leqslant \tau$ for some $\mathcal{D}'$ obtained from $\mathcal{D}$ by at most k modifications, then $J \in \mathbb{A}$.

Proof. Consider the core of algorithm A-FBIMINER in which all pruning is disabled. Then, A-FBIMINER mines all itemsets and returns the set $\mathbb{A}_\tau$ consisting of all J with $|J| > 1$ and $\mathsf{mplift}_k(J, \mathcal{D}) \leqslant \tau$. Since $\mathsf{FBI}(\mathcal{D}', \tau) \subseteq \mathbb{A}_\tau$ the proposition follows.

We next argue that the property holds, even when pruning is enabled. Assume, for the sake of contradiction, that there exists an itemset I with $|I| > 1$ and $\mathsf{lift}(I, \mathcal{D}') \leqslant \tau$, where $\mathcal{D}'$ is a dataset obtained from $\mathcal{D}$ by at most k modifications, yet I is not returned by A-FBIMINER$(\mathcal{D}\downarrow[\emptyset], \emptyset, k, \tau)$. There are only two possibilities why I could not have been returned: Either (i) $\mathsf{mplift}_k(I, \mathcal{D}) > \tau$ or (ii) one of I's subsets was pruned.

Algorithm 11 Mining Almost Forbidden Itemsets

procedure A-FBIMINER($\mathcal{D}\downarrow[P], P \subseteq \mathcal{I}, \tau, k$)
  $\mathbb{A} \leftarrow \emptyset$
  for all $i \in \mathcal{I}$ occuring in $\mathcal{D}\downarrow[P]$ in reverse order **do**
    $I \leftarrow P \cup \{i\}$
    if $k \leqslant \frac{1}{\tau} - 1$ **then**
      if not ISGENERATOR(I) **then**
        continue
      STOREGENERATOR(I)
    if $|I| > 1$ & $\mathsf{freq}(I, \mathcal{D}_r) \leqslant \frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1$ **then**
      if a subset of I has been pruned **then**
        continue
      if $\mathsf{mplift}_k(I, \mathcal{D}) < \tau$ **then**
        $\mathbb{A} = \mathbb{A} \cup I$
      if $|\mathcal{D}|/\tau > \min_{S \subset I} \{ (\mathsf{supp}(S, \mathcal{D}_r) + |\mathcal{D}_{dirty}|) \times$
          $(\mathsf{supp}(I \setminus S, \mathcal{D}_r) + |\mathcal{D}_{dirty}|) \}$ **then**
        continue
    if $\mathsf{supp}(I, \mathcal{D}_r) < \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r} + k) \times \tau} - k$ **then**
      continue
    $\mathcal{D}\downarrow[I] \leftarrow \emptyset$
    for all $j \in \mathcal{I}$ in $\mathcal{D}$ such that $j > i$ **do**
      $C \leftarrow \mathsf{cov}(\{i\}, \mathcal{D}) \cap \mathsf{cov}(\{j\}, \mathcal{D})$
      $J \leftarrow I \cup \{j\}$
      $\mathsf{supp}(J, \mathcal{D}) \leftarrow |C|$
      $\mathsf{supp}(J, \mathcal{D}_r) \leftarrow \mathsf{supp}(J, \mathcal{D}) - \mathsf{supp}(J, \mathcal{D}_{dirty})$
      if $\mathsf{supp}(I, \mathcal{D}) - \mathsf{supp}(J, \mathcal{D}) \geqslant \frac{1}{\tau} - 1 - k$ **then**
        $\mathcal{D}\downarrow[I] \leftarrow \mathcal{D}\downarrow[I] \cup \{(j, C)\}$
    $\mathbb{A} \leftarrow \mathbb{A} \cup$ A-FBIMINER($\mathcal{D}\downarrow[I], I, \tau, k$)
  return $\mathbb{A}$

However, (i) is impossibly since $\mathsf{lift}(I, \mathcal{D}') \leqslant \tau$ implies $\mathsf{mplift}_k(I, \mathcal{D}) \leqslant \tau$. Similarly, (ii) is impossible. Indeed, when a subset of I is pruned, Propositions 14–16 imply that I and all its supersets have $\mathsf{lift}(I, \mathcal{D}') > \tau$, again contradicting the initial assumption that I was τ-forbidden.

We may thus conclude that A-FBIMINER returns (at least) all I such that $|I| > 1$ and $\mathsf{lift}(I, \mathcal{D}') \leqslant \tau$ for all $\mathcal{D}'$ obtained from $\mathcal{D}$ by at most k modification. In other words, the set $\mathbb{A}$ returned by A-FBIMINER satisfies property (†) stated in Proposition 12 when L is taken to be the lift measure. □

7.7 Repair Algorithm

We are finally ready to describe our algorithm REPAIR, shown in Algorithm 12. It takes as input the dirty and clean objects, $\mathcal{D}_{dirty}$ and $\mathcal{D}_r$, respectively, a similarity function sim, a linkage scheme ℓ, the lift threshold τ, and block size r.

The dirty objects $\mathcal{D}_{dirty}$ are (arbitrarily) partitioned in blocks $\mathcal{R}_i$ of size r (line 3). For reasons that will become clear later on, we assume that $\tau < 3/4$.

First, two sets, $\mathcal{D}'$ and $\mathcal{D}''$, are initalized to $\emptyset$ (line 2). The set $\mathcal{D}'$ will consist of the repaired objects while the set $\mathcal{D}''$ will consist of objects in $\mathcal{D}_{\mathsf{dirty}}$ for which no repair can be found. We denote by $\mathcal{D} \oplus \mathcal{D}'$ the set of objects obtained by replacing the objects in $\mathcal{D}_{\mathsf{dirty}}$ by their repairs in $\mathcal{D}'$. By default, the set $\mathcal{D}_r$ is not altered during the entire repair process, ensuring that subsequent repairs are not based on previously imputed values, to avoid the propagation of undesirable repair choices. It would, however, require only a minor change if the user wishes to update $\mathcal{D}_r$ every time an object is repaired.

The algorithm iteratively processes each block $\mathcal{R}_i$. When considering $\mathcal{R}_i$, the set of almost forbidden itemsets $\mathbb{A}_i$ is computed using A-FBIMINER$(\mathcal{D} \oplus \mathcal{D}', \tau, r)$ (line 4), where $\mathcal{D}'$ contains the repaired objects for all previous blocks $\mathcal{R}_j$, $j < i$. After that, the repair process for objects in $\mathcal{R}_i$ depends only on $\mathcal{R}_i$ and $\mathbb{A}_i$.

The core of the repair algorithm is the processing of each dirty object o in $\mathcal{R}_i$, separately. The algorithm consecutively processes the clean objects o_c, and computed a modified version of the dirty object o by means of the procedure MODIFY(o, o_c) (line 8). The resulting object is denoted by o_{rep}. The goal of this is to *cluster* all the clean objects o_c together which lead to the same modification o_{rep}. In our implementation, MODIFY(o, o_c) replaces items (A, v) in o by $(A, o_c[A])$ that occur in the τ-forbidden itemsets covered by o. In other words, those items in o that are part of forbidden itemsets are replaced with the corresponding values from o_c, i.e., only the items that are part of inconsistencies are modified.

After the clusters are computed, we now process each cluster in order of their similarity to the dirty object o (**for** loop lines 11–15). To compute the similarity between an object and a cluster of objects, our implementation supports three linkage schemes ℓ inspired by hierarchical clustering:

(i) Single Linkage[1], equal to the highest similarity of o with any object in the cluster;

(ii) Complete Linkage, equal to the lowest similarity of o with any object in the cluster; and

(iii) Mean Linkage, equal to the mean over the similarity of o to all objects in the cluster.

By using the most similar objects to produce *candidate repairs*, we heuristically attempt to minimize the difference between $\mathcal{D}'$ and $\mathcal{D}$. This approach is in line with the commonly used *hot deck imputation* in statistical survey editing [59]. Additionally, the use of different linkage models allows us to take into account information about *all* objects that suggest the same modification. Our experiments show that Mean Linkage is typically more robust to the presence of dirtyness/noise in $\mathcal{D}_r$.

[1]This is equivalent to the nearest neighbour method we initially used in Rammelaere et al. [95].

Algorithm 12 Repairing dirty objects

```
1:  procedure REPAIR(D_dirty, D_r, sim, ℓτ, r)
2:      D' := ∅; D'' = ∅
3:      for all R_i ∈ BLOCKS(D_dirty, r) do
4:          A_i = A-FBIMINER(D ⊕ D', τ, r)
5:          Clust ← [ _ ]
6:          for all o ∈ R_i do
7:              for all o_c ∈ D_r in sim(o_c, o) desc. order do
8:                  o_rep = MODIFY(o, o_c)
9:                  Clust[o_rep] ← Clust[o_rep] ∪ {o_c}
10:             success := False
11:             for all (o_rep, cl) ∈ Clust in ℓ(cl, o, sim) desc. order do
12:                 if SAFE(o, o_rep, A_i) then
13:                     success := True
14:                     D' := D' ∪ o_rep
15:                     break
16:             if not success then D'' = D'' ∪ {o}
17:         return (D', D'')
```

It remains to ensure that if we augment $\mathcal{D}'$ with the candidate repair o_{rep} associated with the most similar cluster (line 14) , then $\mathcal{D} \oplus \mathcal{D}'$ is "cleaner" than $\mathcal{D}$. That is, no new forbidden itemsets are introduced. Furthermore, if all objects in $\mathcal{D}_{\mathsf{dirty}}$ are successfully repaired, then no forbidden itemsets are found in $\mathcal{D} \oplus \mathcal{D}'$. That is, $\mathcal{D} \oplus \mathcal{D}'$ is clean. This is guaranteed by the safety check (procedure SAFE, line 12) in the algorithm. If the candidate repair is safe with respect to the almost forbidden itemsets in $\mathbb{A}_i$, then it is added to the set $\mathcal{D}'$ (line 14). Otherwise, the next candidate cluster is considered (**for** loop lines 11–15) until a repair is found (line 13) or all candidate repairs have been rejected. In this case, o is added to the set of unrepaired objects $\mathcal{D}''$ (line 16) for user inspection.

In the remainder of this section we describe the procedure SAFE in detail. Consider a run of the algorithm in which objects in $\mathcal{D}_{\mathsf{dirty}}$ are repaired. Assume that the dirty objects in $\mathcal{D}_{\mathsf{dirty}}$ are considered in the following order: $o_1, o_2, \ldots, o_k$. Let $\mathcal{D}'_0 = \mathcal{D}$, and denote by $\mathcal{D}'_i$ the modified instance $\mathcal{D}'_{i-1} \oplus o_{rep,i}$ in which $o_i \in \mathcal{D}_{\mathsf{dirty}}$ is repaired. In case all dirty objects can be repaired, the final set $\mathcal{D}'_k$ is to be a repair of $\mathcal{D}$, i.e., $\mathsf{FBI}(\mathcal{D}'_k, \tau) = \emptyset$. Similarly, we let $\mathcal{D}_0 = \mathcal{D}$, $\mathcal{D}_i = \mathcal{D}_{i-1} \oplus \mathcal{D}_{rep,i}$ to denote the repair obtained after processing i blocks in the partitioning of $\mathcal{D}_{\mathsf{dirty}}$. If there are ℓ blocks, then $\mathcal{D}_\ell$ is a repair of $\mathcal{D}$.

Suppose that the algorithm just finished processing block $\mathcal{R}_i$, resulting in the partial repair $\mathcal{D}_i$ and almost forbidden itemsets $\mathbb{A}_i =$ A-FBIMINER$(\mathcal{D}_i, \tau, r)$. Suppose that objects $o_j, \ldots, o_{j+r-1}$ constitute the next block $\mathcal{R}_{i+1}$ of dirty objects, where $j = r \cdot i + 1$.

The procedure SAFE is based on four conditions that together ensure that for each $k \in [ir+1, (i+1)r]$, $\mathsf{FBI}(\mathcal{D}'_k, \tau) \subseteq \mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$ and thus $\mathcal{D}'_k$ can be regarded as being cleaner than or as clean as $\mathcal{D}'_{k-1}$. We next describe these conditions. To begin with, let $\mathbb{P}_i := \{J \in \mathbb{A}_i \mid \mathsf{lift}(J, \mathcal{D}_i) > \tau\}$. We first guaran-

tee that r successive and succesful modifications to $\mathcal{D}_i$ can never decrease the lift of itemsets in $\mathbb{P}_i$. Consider object $o_k = \langle \mathsf{tid}, I \rangle$ in $\mathcal{D}'_{k-1}$ and its candidate modification $o_{rep,k} = \langle \mathsf{tid}, I' \rangle$ in $\mathcal{D}'_k$. Define $\mathsf{covitems}(I, X)$ for an itemset I and set of itemsets X to be those itemsets $J \in X$ such that $J \subseteq I$. Similarly, $\mathsf{itemscov}(I, X)$ is the set of itemsets J in X such that $I \subseteq J$.

Proposition 19. Consider object $o_k = \langle \mathsf{tid}, I \rangle$ in $\mathcal{D}'_{k-1}$ and its candidate modification $o_{rep,k} = \langle \mathsf{tid}, I' \rangle$ in $\mathcal{D}'_k$. Let $\tau < 3/4$. If the following conditions are satisfied

(a) $\mathsf{covitems}(I, \mathbb{P}_i) \subseteq \mathsf{covitems}(I', \mathbb{P}_i)$; and

(b) for each $a \in I' \setminus I$,

$$\mathsf{itemscov}(\{a\}, \mathbb{P}_i) \subseteq \mathsf{covitems}(I', \mathbb{P}_i),$$

then for each $J \in \mathbb{P}_i$, we have that $\mathsf{lift}(J, \mathcal{D}'_{k-1}) > \tau$ implies $\mathsf{lift}(J, \mathcal{D}'_k) > \tau$.

Proof. We only have to consider the case when $\mathsf{lift}(J, \mathcal{D}'_{k-1}) > \mathsf{lift}(J, \mathcal{D}'_k)$. By assumption and by the definition of lift, we have that

$$\tau < \mathsf{lift}(J, \mathcal{D}'_{k-1}) = \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}'_{k-1})}{\mathsf{supp}(S, \mathcal{D}'_{k-1}) \times \mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1})}$$

for some $S \subset J$. If J is supported by neither o_i (in $\mathcal{D}'_{k-1}$) nor $o_{rep,i}$ (in $\mathcal{D}'_k$), then $\mathsf{supp}(J, \mathcal{D}'_{k-1}) = \mathsf{supp}(J, \mathcal{D}'_k)$. Otherwise, condition (a) implies that if J is supported by o_i, and hence $J \in \mathsf{covitems}(I, \mathbb{P}_i)$, then also $J \in \mathsf{covitems}(I', \mathbb{P}_i)$. Hence, J is also supported by $o_{rep,i}$. From this we may conclude that $\mathsf{supp}(J, \mathcal{D}'_{k-1}) \leqslant \mathsf{supp}(J, \mathcal{D}'_k)$.

Assume that $\mathsf{lift}(J, \mathcal{D}'_{k-1}) > \mathsf{lift}(J, \mathcal{D}'_k)$. Given that $\mathsf{supp}(J, \mathcal{D}'_{k-1}) \leqslant \mathsf{supp}(J, \mathcal{D}'_k)$, it must be the case that $\mathsf{supp}(S, \mathcal{D}'_k) \times \mathsf{supp}(J \setminus S, \mathcal{D}'_k) > \mathsf{supp}(S, \mathcal{D}'_{k-1}) \times \mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1})$. Since only one modification is considered at a time, the maximal value of $\mathsf{supp}(S, \mathcal{D}'_k) \times \mathsf{supp}(J \setminus S, \mathcal{D}'_k)$ is given by $(\mathsf{supp}(S, \mathcal{D}'_{k-1}) + 1) \times (\mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1}) + 1)$.

This in turn can only occur when S and $J \setminus S$ contain items in $I' \setminus I$ (otherwise, their supports would be the same as in $\mathcal{D}'_{k-1}$). For example, let $a \in I' \setminus I$ such that $a \in S$. Then, $a \in J$ and hence $J \in \mathsf{itemscov}(\{a\}, \mathbb{P}_i)$. Hence, condition (b) implies that J is supported by $o_{rep,i}$. In addition, since S and $J \setminus S$ contain items in $I' \setminus I$, o_i cannot support S and $J \setminus S$. Hence, o_i also did not support J. This implies that $\mathsf{supp}(J, \mathcal{D}'_k) = \mathsf{supp}(J, \mathcal{D}'_{k-1}) + 1$. The proposition now follows from the following implication, whose proof is deferred to Appendix C.2: For $\tau < 3/4$,

$$\tau < \frac{|\mathcal{D}| \times \mathsf{supp}(J, \mathcal{D}'_{k-1})}{\mathsf{supp}(S, \mathcal{D}'_{k-1}) \times \mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1})}$$

implies that

$$\tau < \frac{|D| \times (\mathsf{supp}(J, \mathcal{D}'_{k-1}) + 1)}{(\mathsf{supp}(S, \mathcal{D}'_{k-1}) + 1) \times (\mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1}) + 1)},$$

which in turn is smaller than or equal to $\mathsf{lift}(J, \mathcal{D}'_k)$. This concludes the proof of the proposition. □

As shown in Algorithm 13, the procedure SAFE will reject the candidate modification $o_{rep,k}$ when conditions (a) and (b) are not satisfied (lines 3–7). In addition, the procedure SAFE performs two additional checks. In particular, let $\mathbb{S}0_i$ be the set of itemsets in $\mathbb{A}_i$ that do not have support in $\mathcal{D}_i$. Then, $o_{rep,k}$ is rejected by procedure SAFE (lines 8, 9) when the following condition is not satisfied:

(c) $\mathsf{covitems}(I', \mathbb{S}0_i) = \emptyset$.

That is, for $o_{rep,k} = \langle \mathsf{tid}, I' \rangle$ to be a good repair, I' is not allowed to contain almost forbidden itemsets in $\mathbb{A}_i$ that do not have support in $\mathcal{D}_i$. Similarly, if we let $\mathbb{O}_i$ be the set of itemsets in $\mathbb{A}_i$ that are not forbidden in $\mathcal{D}_i$, then $o_{rep,k}$ is rejected by procedure SAFE (lines 10,11) when the following condition is not satisfied:

(d) $\mathsf{covitems}(I', \mathbb{O}_i) = \emptyset$.

That is, for $o_{rep,k} = \langle \mathsf{tid}, I' \rangle$ to be a good repair, I' is not allowed to contain almost forbidden itemsets in $\mathbb{A}_i$ that were forbidden in $\mathcal{D}_i$.

We next show correctness of the procedure SAFE. Let $\mathcal{D}_i$, $\mathbb{A}_i$, $\mathcal{D}'_k$, $\mathcal{D}'_{k-1}$, o_k and $o_{rep,k}$ as before.

Proposition 20. If the procedure $\textsc{Safe}(o_k, o_{rep,k}, \mathbb{A}_i)$ returns `True`, then

$$\mathsf{FBI}(\mathcal{D}'_k, \tau) \subseteq \mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$$

Proof. We show this by contradiction. Suppose that there exists an itemset $J \in \mathsf{FBI}(\mathcal{D}'_k, \tau)$ which is not in $\mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$. Since $\mathcal{D}'_k$ is at most r modifications removed from $\mathcal{D}_i$, we have that $J \in \mathbb{A}_i$. There are only two possible reasons why $J \notin \mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$. The first reason might be that $\mathsf{supp}(J, \mathcal{D}'_{k-1}) = 0$. We show that this implies that also $\mathsf{supp}(J, \mathcal{D}_i) = 0$. This in turn implies that $J \in \mathbb{S}0_i$. Together with the assumptions that $J \in \mathsf{FBI}(\mathcal{D}'_k, \tau)$ and $\mathsf{supp}(J, \mathcal{D}'_{k-1}) = 0$, we can infer that $J \in \mathsf{covitems}(I', \mathbb{S}0_i)$. This contradicts that condition (c) is satisfied for $o_{rep,k}$ (recall that $\textsc{Safe}(o_k, o_{rep,k}, \mathbb{A}_i)$ returns `True`, by assumption). We still need to show that $\mathsf{supp}(J, \mathcal{D}'_{k-1}) = 0$ implies $\mathsf{supp}(J, \mathcal{D}_i) = 0$. Suppose that $\mathsf{supp}(J, \mathcal{D}_i) > 0$. If $\mathsf{lift}(J, \mathcal{D}_i) \leqslant \tau$, then $J \in \mathbb{O}_i$ and hence $J \in \mathsf{covitems}(I', \mathbb{O}_i)$. We again reach a contradiction, because condition (d) is satisfied for $o_{rep,k}$. Hence, $\mathsf{lift}(J, \mathcal{D}_i) > \tau$. However, since conditions (a) and (b) are satisfied, Proposition 19 implies that $\mathsf{lift}(J, \mathcal{D}'_k) > \tau$. This contradicts our assumption that $J \in \mathsf{FBI}(\mathcal{D}'_k, \tau)$.

Hence, $J \notin \mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$ implies that $\mathsf{lift}(J, \mathcal{D}'_{k-1}) > \tau$. Indeed, this is the second reason why J might not belong to $\mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$. Again, since conditions (a) and (b) are satisfied, Proposition 19 implies that $\mathsf{lift}(J, \mathcal{D}'_k) > \tau$. This again contradicts our assumption that $J \in \mathsf{FBI}(\mathcal{D}'_k, \tau)$.

In other words, if $J \in \mathsf{FBI}(\mathcal{D}'_k, \tau)$ then we must have that $J \in \mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$, as desired. □

We observe that procedure SAFE may reject candidate modifications for which $\mathsf{FBI}(\mathcal{D}'_k, \tau) \subseteq \mathsf{FBI}(\mathcal{D}'_{k-1}, \tau)$ holds. Indeed, the conditions (a)–(d) are only sufficient conditions. The reason for allowing such wrong rejections is efficiency. Indeed, each condition only requires information about dirty objects

Algorithm 13 Checking whether a repair is safe

```
1: procedure SAFE(o_k = ⟨tid, I⟩, o_rep,k = ⟨tid, I'⟩, 𝔸_i)
2:     Compute ℙ_i, 𝕆_i and 𝕊0_i
3:     if covitems(I, ℙ_i) ⊈ covitems(I', ℙ_i) then
4:         return False
5:     for all a ∈ I' \ I do
6:         if itemscov({a}, ℙ_i) ⊈ covitems(I', ℙ_i) then
7:             return False
8:     if covitems(I', 𝕊0_i) ≠ ∅ then
9:         return False
10:    if covitems(I', 𝕆_i) ≠ ∅ then
11:        return False
12:    return True
```

and candidate modifications, and the set $\mathbb{A}_i$ of almost forbidden itemsets, which is already in place. More exact versions of the procedure SAFE could be considered, at the cost of time-consuming lift computations.

A direct consequence of Proposition 20 is that when a dirty object o_k is repaired into object $o_{rep,k}$, then $o_{rep,k}$ does not contain any forbidden itemsets from $\mathsf{FBI}(\mathcal{D}'_k, \tau)$. Indeed, suppose that o_{rep} supports an itemset $J \in \mathsf{FBI}(\mathcal{D}'_k, \tau)$. Then the proposition implies that $J \in \mathsf{FBI}(\mathcal{D}, \tau)$. Condition (d), however, ensures that $o_{rep,k}$ is rejected whenever an "old" forbidden itemset is supported.

Given this, we conclude this section by showing the correctness of the REPAIR algorithm.

Proposition 21. Let $\mathcal{D}$ be a dataset, τ a maximum lift threshold, sim a similarity function, and let $\mathcal{D}_{dirty}$ and $\mathcal{D}_{clean}$ denote the dirty and clean parts of $\mathcal{D}$, respectively. Algorithm REPAIR($\mathcal{D}_{dirty}$, $\mathcal{D}_{clean}$, sim, τ) returns sets $\mathcal{D}'$ and $\mathcal{D}''$ of repaired and unrepairable objects. If the repair was successful, i.e., $\mathcal{D}'' = \emptyset$, then $\mathsf{FBI}(\mathcal{D}_{clean} \cup \mathcal{D}', \tau) = \emptyset$.

Proof. For the sake of contradiction, suppose that there exists a τ-forbidden itemset J in $\mathsf{FBI}(\mathcal{D}_{clean} \cup \mathcal{D}', \tau)$.

Observe that no object in D_r supports a τ-forbidden itemset in $\mathsf{FBI}(\mathcal{D}, \tau)$. By Proposition 20, no modification made by algorithm REPAIR can create new forbidden itemsets. Since REPAIR does not make modifications to clean objects, no object in $\mathcal{D}_{clean}$ supports a τ-forbidden itemset in $\mathsf{FBI}(\mathcal{D}_{clean} \cup \mathcal{D}', \tau)$ either.

Suppose that J is supported by an object $o_i \in \mathcal{D}'$. Then, Proposition 20 implies that $J \in \mathsf{FBI}(\mathcal{D}'_k, \tau)$. We have just observed that accepted repairs cannot support such forbidden itemsets. Consequently, no τ-forbidden itemset J in $\mathsf{FBI}(\mathcal{D}_{clean} \cup \mathcal{D}', \tau)$ can exist and therefore $\mathcal{D}_{clean} \cup \mathcal{D}'$ is indeed clean. □

As a final remark, we would like to point out that the assumption that $\tau < 3/4$ is not a restriction in practice. As we will see in the experiments, reasonable values for τ are typically around 0.1 or less.

Table 7.1: Statistics of the UCI datasets used in the experiments. We report the number of objects, distinct items, and attributes.

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

7.8 Experiments

The experiments were conducted on the same datasets as in the previous chapter, six datasets from the UCI repository [39] and the synthetic Soccer dataset [2]. The statistics of these datasets are repeated in Table 7.1. The algorithms have been implemented in C++, the source code and used datasets are available for research purposes [91]. The program has been tested on an Intel Xeon Processor (2.9GHZ) with 32GB of memory running Linux. Our algorithms run in main memory. Reported runtimes are always an average over five independent runs.

7.8.1 Performance of A-FBIMiner

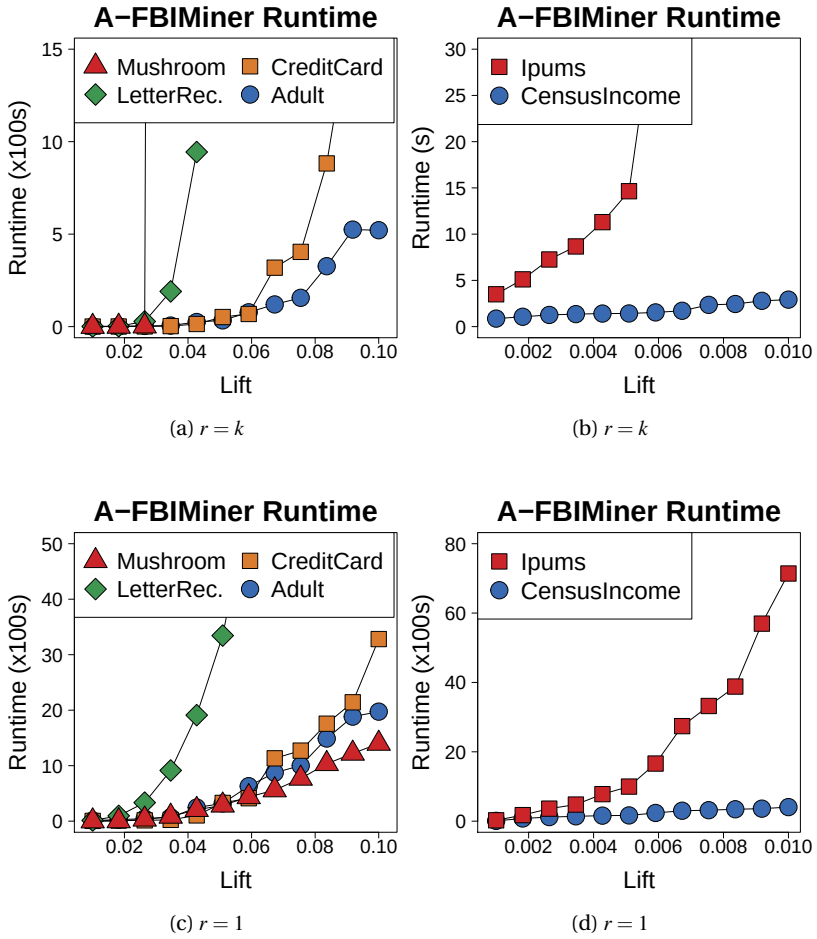
We first investigate the discovery of almost forbidden itemsets, which is the most computationally expensive part in our methodology. Recall that the runtime of A-FBIMINER depends both on the lift threshold τ and the number of dirty objects as discovered by FBIMiner. Since a larger τ automatically entails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

For each dataset and each τ, we first run algorithm FBI-MINER to obtain the forbidden itemsets. Let k denote the number of dirty objects found. We then run algorithm A-FBIMINER a number of times with block size r, indicating the number of dirty objects to be repaired at once, until all k objects have been repaired. We first consider only the extreme cases of the block size, i.e., $r = 1$ and $r = k$. Experiments with other blocksizes are discussed in the next section. The obtained runtimes are shown in Figure 7.2.

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

Similar to FBIMINER, we are also interested in the number of itemsets returned by A-FBIMINER, and the number of objects containing such sets.

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



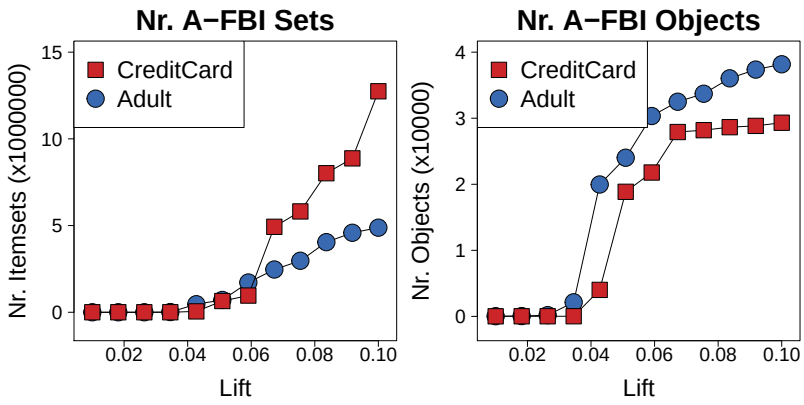
(a) $r = k$ (b) $r = k$

(c) $r = 1$ (d) $r = 1$

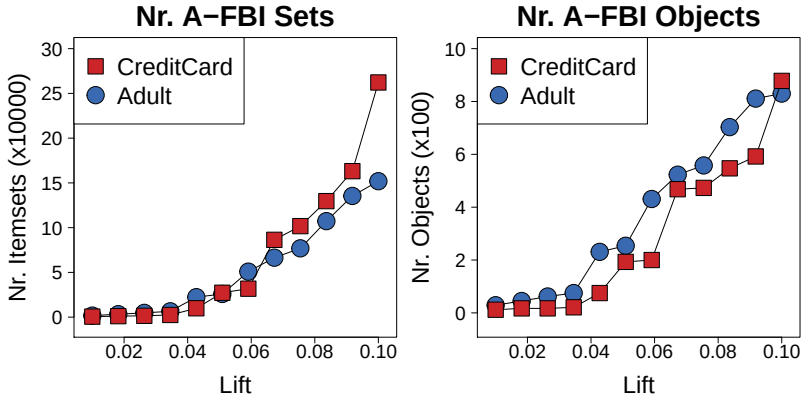
Figure 7.2: Runtime of A-FBIMINER in function of maximum lift threshold τ, for block sizes $r = k$ and $r = 1$.

We perform this analysis, and compare between block sizes $r = 1$ and $r = k$, on the Adult and CreditCard datasets. These datasets permit us to investigate the entire τ-range, unlike the Mushroom and LetterRecognition datasets on which the mining of almost forbidden itemsets quickly becomes infeasible. For block size $r = 1$, the obtained runtime is then a total over k runs. Results are shown in Figure 7.3.

The obtained results show that the block size has a considerable impact on the number of almost forbidden itemsets, which in turn affect repairability. Indeed, when using block size $r = k$, we see that eventually every object in the CreditCard dataset contains at least one almost forbidden itemset, similarly for the Adult dataset. The number of almost forbidden itemsets is very large. When using block size $r = 1$, the number of almost forbidden itemsets is reduced by approximately a factor 100, and the number of objects containing an almost forbidden itemset is a moderate percentage of the dataset.



(a) $r = k$ (b) $r = k$

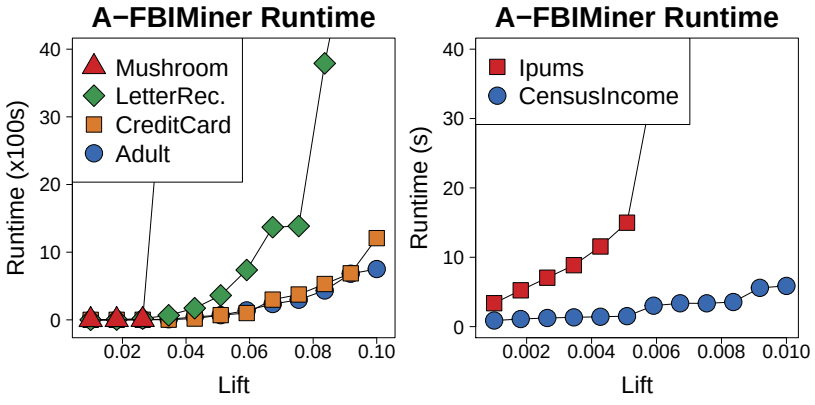


(c) $r = 1$ (d) $r = 1$

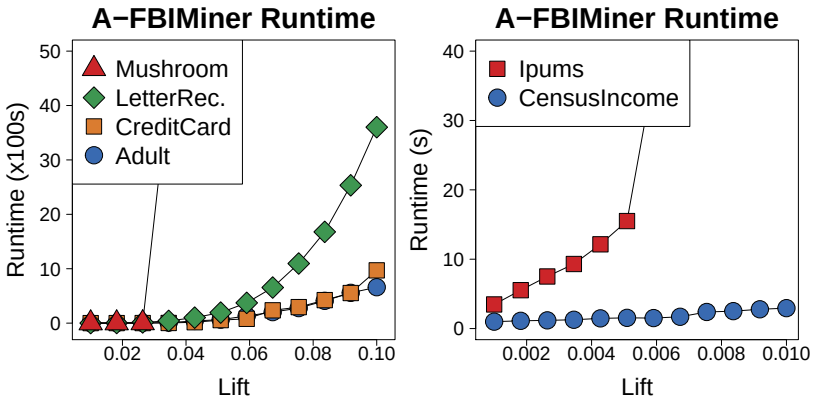
Figure 7.3: Number of discovered Almost Forbidden Itemsets and number of objects with an Almost Forbidden Itemset, for block sizes $r = k$ and $r = 1$.

Block Size

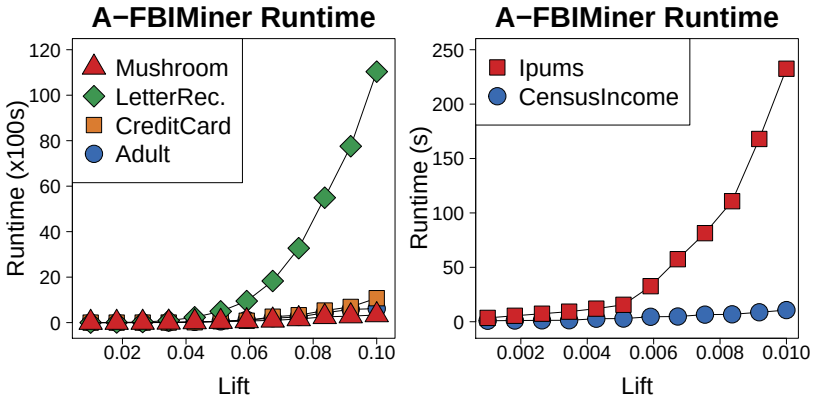
Next, we focus on the optimal block size r. As outlined in Section 7.6, we can identify the quantities $\frac{1}{\tau} - 1$ and $\frac{|\mathcal{D}|}{(\sigma^{max}_{l,\mathcal{D}_r}+r)\times\tau} - 1$ as the maximal block sizes for which Proposition 14 and Proposition 15, respectively, are still applicable. For block sizes less than k, we compute the runtime as a total over a number of consecutive runs of A-FBIMINER, until all k dirty objects are "covered" by the algorithm. Figures 7.4a–7.4d display the obtained runtimes using these block sizes. Note that the chosen values for r are τ-dependent. Consequently, for every τ-value, a different number of dirty objects is obtained and partitioned into blocks of a *different* absolute size. As expected, the runtimes are lower than for $k = 1$, while feasibility is better than for $r = k$, proving that the right block size indeed improves the overall performance of the algorithm. Runtime on the LetterRecognition dataset naturally suffers from the exponential increase in the number of dirty objects on that dataset. The Mushroom and Ipums datasets are still problematic: the number of attributes leads to a deep search tree, and pruning power is too limited.



(a) $r = \frac{1}{\tau} - 1$ (b) $r = \frac{1}{\tau} - 1$



(c) $r = \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau} - 1$ (d) $r = \frac{|\mathcal{D}|}{(\sigma^{max}_{I,\mathcal{D}_r}+r)\times\tau} - 1$



(e) $r = \frac{1}{2\times\tau}$ (f) $r = \frac{1}{2\times\tau}$

Figure 7.4: Runtime of A-FBIMINER in function of maximum lift threshold τ, for various block sizes r.

As an alternative, we consider the block size $r = \frac{1}{2\tau}$, the halfway point between $r = 1$ and $r = \frac{1}{\tau} - 1$. Figure 7.4e–7.4f shows that this block size provides sufficient pruning power for the Mushroom dataset, and indeed outperforms all other considered sizes over the entire τ-range. For higher τ-values, the algorithm still struggles on the Ipums dataset, its high number of items proving problematic. On the other datasets, A-FBIMINER is fast for low values of τ, and feasible across the considered τ-range.

7.8.2 Quality of Repairs

We now turn our attention towards the quality of the repairs made by our algorithm. We first examine whether cleaning the forbidden itemsets brings the dirty data closer to the ground truth. For this, we again make use of the Soccer dataset with inserted violations, since this is the only dataset on which we have the ground truth. Indeed, since the other datasets are already dirty to start with, the repair algorithm is affected. For instance, the forbidden itemsets already present in the data (before inserting violations) mark certain objects as dirty, removing them from consideration as donors in the repair algorithm. As such, we cannot evaluate repairs fairly on such datasets. At the end of this section, we detail the similarity function used; the distance between two objects is computed as $1 - \mathsf{sim}$.

In Figure 7.5, we display the results on the Soccer dataset, with errors from 1 and 50 CFDs. The block size $r = \frac{1}{2\tau}$ was chosen, as described in the previous section. As the value of τ increases, the repaired data becomes similar to the ground truth. Clearly, this is contingent on the discovered forbidden itemsets having a high precision; therefore, we only consider the τ values 0.05, $0,10$, and 0.15, as it was shown in the previous chapter that our precision drops significantly for higher values of τ. For a τ-value of 0.05, the Single Linkage scheme performs worse on the version of the dataset with errors from 1 CFD, while Complete Linkage performs slightly worse on the version with errors from 50 CFDs. The Mean Linkage scheme performs well in every situation. For higher τ, the repair schemes obtain similar results. This happens because, for smaller τ, certain erroneous objects are not detected as such. Consequently, these objects are part of the different clusters for repairing, and influence the distances to the object to be repaired. In other words, for lower τ, there is a higher risk of making a modification based on an object which is in fact erroneous, but not detected as such by our method. We conclude that the Mean Linkage scheme is generally the most robust, but the differences in performance are limited.

For the other datasets, on which we cannot evaluate closeness to the ground truth, we instead investigate the minimality of the repairs we obtain. In Table 7.2, the minimal and maximal similarity between a dirty object and its repair given by algorithm REPAIR (within a given τ range). Here, a similarity value of 1 indicates identical objects. Again, the block size $r = \frac{1}{2\tau}$ was chosen. The obtained repairs consistently have a high similarity in the given τ-range, indicating that our repair method makes very limited alterations to the dirty objects in order to make them clean. Obviously, it is not desirable for a repair algorithm to substantially distort the input database, especially

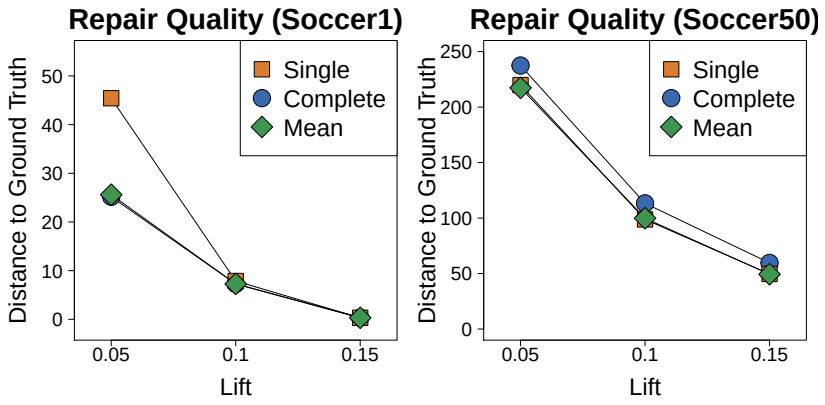


Figure 7.5: Distance between repaired dataset and ground truth, using three different repair schemes, in function of threshold τ. Results were obtained on the Soccer dataset with errors generated by 1 and 50 cCFDs.

Table 7.2: Average quality of repairs.

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

as we consider low error rate data. We conclude that our repairs are of high quality.

We also report the number of objects that could not be repaired at the highest τ-value, denoted by $|\mathcal{D}''|$. For Adult, CensusIncome, CreditCard and LetterRecognition, only a few objects are unrepairable and this occurs only for high values of τ. A higher number of unrepairable objects is encountered for the Ipums and Mushroom datasets. This seems to suggest that a higher number of attributes causes problems for repairing.

To illustrate the effects of our repair algorithm, we show example repairs obtained on the Adult dataset in Figure 7.6, for $\tau = 0.01$ and using Single Linkage. The five dirty objects shown contain the forbidden itemsets (SEX=MALE, RELATION=WIFE), (RELATION=NOT-IN-FAMILY, MARITAL=MARRIED), (RELATION=HUSBAND, MARITAL=MARRIED), and/or (MARITAL-STATUS=MARRIED, AGE=$<$18). In the first dirty object, we see that our repair algorithm effectively manages to remove both forbidden itemsets from this object by changing only the Age value. Moreover, all modifications seem sensible.

As a final comment on the topic of repair quality, we detail the similarity measure sim used in the REPAIR algorithm. While similarity and distance

...	Age	Marital Status	Relationship	Sex	...
	<18	Married-Civ.	Husband	Male	
	>50	Married-Civ.	Husband	Male	
	<18	Married-Civ.	Own Child	Female	
	<18	Never Married	Own Child	Female	
	22-30	Married-Civ.	Not in Family	Male	
	22-30	Never Married	Own Child	Male	
	22-30	Married-Civ.	Wife	Male	
	22-30	Married-Civ.	Wife	Female	
	31-50	Married-Civ.	Wife	Male	
	31-50	Married-Civ.	Husband	Male	

Figure 7.6: Example repairs on Adult dataset.

measures are well-studied for numerical data, it is often more challenging to measure similarity of categorical data. Boriah et al. [20] give an overview of similarity and distance measures for categorical data, and distinguish three types of measures, based on whether they assign different weights to either matches or mismatches between values, or to both. For our algorithm, we make use of the lin-similarity measure which weights both matches and mismatches based on the frequency of the actual values. For example in the context of census data, a match or mismatch in gender would be more influential than a match or mismatch in the age category, since the domain of the Gender attribute is much smaller. Of course, any other similarity measure could be used instead.

Definition 29 (Lin-similarity measure)**.**

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

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

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

7.8.3 Repair Runtime

To conclude the experimental evaluation of our method, we investigate the runtime of the repair algorithm. We also make a comparison between sequential repairing and a parallel implementation using OpenMP. The results are shown in Figure 7.7. The time required to repair is mostly dependent on the number of dirty objects, while the time per object depends on the number of clean objects, i.e., objects for which the similarity to the dirty object needs to be computed. As such, the runtime plots are similar in shape to the plots in Figure 6.5 of the previous chapter, i.e., the number of dirty objects. We see that repairing in parallel provides a considerable speedup. Note that the repair algorithm itself is independent of τ, which only affects FBIMINER and A-FBIMINER.

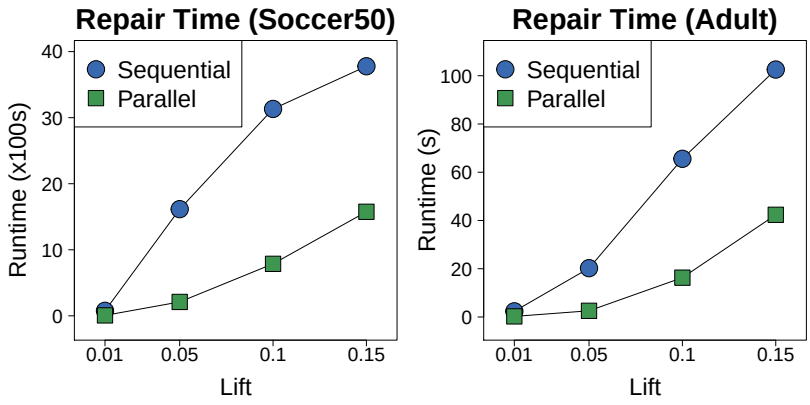


Figure 7.7: Comparison of sequential and paraller Repair runtime, on Soccer dataset with errors for 50 cCFDs and Adult.

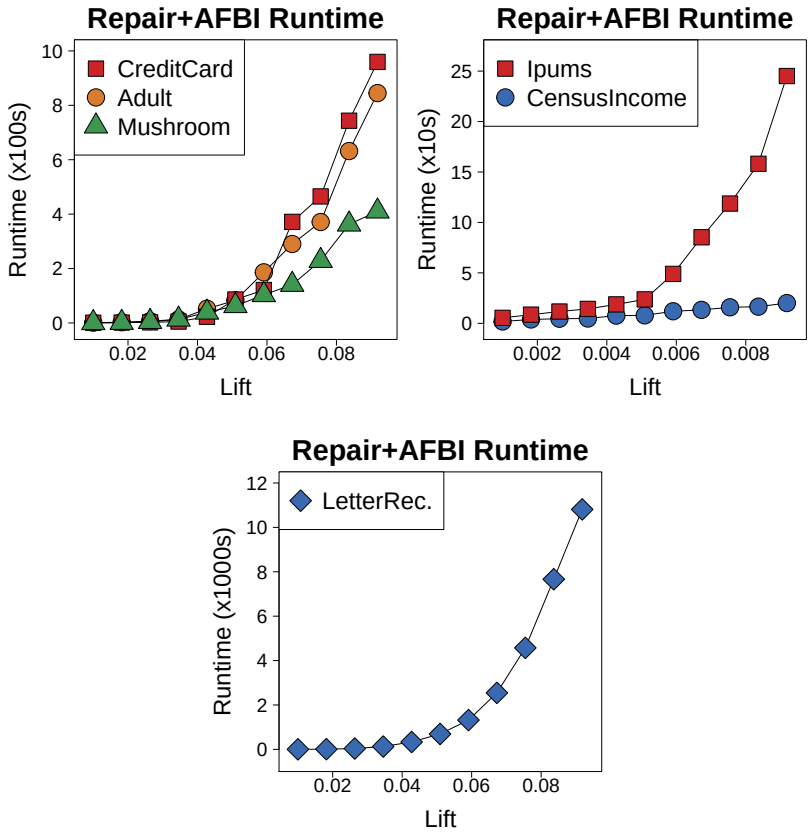


Figure 7.8: Runtime of repairing, including the mining of almost forbidden itemsets with blocksize $1/2\tau$, on various datasets.

In the previous experiment, the runtime for algorithm REPAIR was shown in isolation. However, in order to perform correct repairs, it is also necessary to compute almost forbidden itemsets for every block of dirty objects. In Figure 7.8, we report runtimes which include the runtime of A-FBIMINER. While the runtimes increase noticeably, the overall scalability of repairing on the various datasets is still retained. Only on the LetterRecognition dataset, the runtimes of A-FBIMINER rise very strongly for larger values of τ, as already evidenced by the A-FBIMINER runtime experiments.

7.9 Conclusion

In this chapter, we have address the problem of repairing data under the dynamic notion of cleanliness, introduced in the previous chapter. We have developed a flexible repair algorithm based on imputating values from similar, but clean, objects. Formal proofs guarantee that after repairs, no new inconsistencies can be found.

Furthermore, we have introduced the concept of almost forbidden itemsets, which are necessary for the efficiency of our method. By first mining almost forbidden itemsets, we can assure that no itemsets become forbidden during a repair. This is an essential ingredient in our dynamic notion of data quality. Crucial here are our pruning strategies for mining almost forbidden itemsets, and our guidelines for repairing in batches of suitable size.

We have shown experimentally that our repairs are of high quality, and bring dirty data close to the ground truth. Moreover, the use of almost forbidden itemsets enables repairs to be performed in parallel, further improving runtimes.

CHAPTER 8

Conclusion

Dirty data is a significant problem for companies and analysts alike. Within the field of data cleaning research, many techniques focus on a constraint-based paradigm: rules in some logical formalism indicate which parts of the data are dirty, and repair algorithms subsequently modify the data such that all constraints are satisfied. Throughout this dissertation, we have considered the problem of constraint discovery from different angles, leveraging work in the area of pattern mining. To conclude the dissertation, we summarize the contributions made in the presented work, and discuss possible avenues for future research in the area of constraint discovery for data cleaning.

8.1 Main Contributions

- In **Chapter 4**, we revisited the discovery of *conditional functional dependencies (CFDs)*. These constraints are widely used in data cleaning, since they are easy to understand, yet more expressive than standard functional dependencies (FD) or association rules. We have recast CFD discovery as an explicit combination of FD discovery and itemset mining, and identified three different ways in which these can be combined. As such, CFD discovery can benefit from advances in both disciplines. We further generalized the pruning of redundant CFDs, allowing different search strategies to be used instead of the traditional breadth-first search. Experiments show that our new methodologies, as well as the correct choice of search strategy, can provide a substantial performance increase when compared to the state of the art CFD discovery algorithm CTane.

- In **Chapter 5**, we considered the problem of finding CFDs that are *valid and useful for repairing*, based on user interaction. Instead of relying on the user to (in)validate rules, we presented a method where only positive feedback is required: The user manually repairs a small part of the data, and our algorithm discovers a CFD that *explains* this partial repair. We introduce a scoring function for such explanations, and present an efficient on-demand algorithm, XPLODE, for discovering the best explanation given a set of modifications. Our experiments 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.

- In **Chapter 6**, we introduced a *dynamic notion of data quality*, where we consider a given database to be clean if a constraint discovery algorithm does not detect any violated constraints on that data. We phrased this notion in full generality, before studying the problem concretely for an easy to understand constraint language, which we call *forbidden itemsets*. Such itemsets have *low lift*, and we extensively motivated their use for detecting erroneous values in data. To discover forbidden itemsets, we have introduced an algorithm called FBIMiner, and derived properties of the lift measure to provide strong pruning. In the experiments, we showed that forbidden itemsets can discover errors with high precision, and FBIMiner obtains a good runtime.

- In **Chapter 7**, we presented a method for *repairing* data in the context of forbidden itemsets, under our dynamic notion of data quality. To avoid having to recompute forbidden itemsets for every candidate repair, we introduced *almost forbidden itemsets*, which are itemsets that could become forbidden after a set number of modifications by the repair algorithm. As such, we can compute up front enough information to guarantee the cleanliness of repairs. To further improve runtime, we proposed repairing in batches of suitable size. The repair algorithm itself modifies dirty objects by taking suggestions from clean parts of the data, which are clustered together, and similarity to the clusters is computed flexibly using linkage schemes from hierarchical clustering. We formally proved the correctness of the entire repair method. In the experiments, we evaluated the runtime of mining almost forbidden itemsets. We further showed that the runtime of repairing can be improved by repairing in parallel, and that our repairs bring the dirty data closer to the ground truth.

8.2 Outlook

In the conclusions of Chapters 4–7, we have discussed very specific directions for extending the work presented in those sections. As we have come to the end of this dissertation, we now take the time to outline some general paths in which data mining and machine learning techniques can further push the field of constraint-based data quality.

Cleaning other types of data. Throughout the dissertation, we have focused mostly on categorical data in a tabular format. Numerical data can be incorporated into these methods by discretization. However, many other types of data exist, such as multi-table relational data, sequential data, graph data, or live data in a streaming format. Constraint-based cleaning on such data has received comparably little attention. It would be interesting to see how our methods, or similar approaches, could be adapted to scenarios with such data types. For example, the forbidden itemsets formalism could be applied to other data, by devising a suitable interestingness measure for detecting errors in such data.

Leveraging recommendation systems for interactive error discovery. In Chapter 5, we considered user interaction in the form of manual repairs made by a user. Based on these interactions, the XPLODE algorithm discovered constraints that align with those repairs. Conceptually, this is similar to the problem setting in recommendation systems, where the quintessential example is a user buying items, and an algorithm recommending items that are "also interesting" to the user. When translated to a data cleaning setting, this could be interpreted as a user marking errors, and the algorithm suggesting items that are "also errors". In the field of recommendation systems, user friendliness has been considered extensively, and such systems are designed for rapid and flexible interaction, two considerations which are also crucial in user interactive data cleaning. Furthermore, by considering a constraint in general as "a rule that indicated errors", error detection based on recommendation systems can be agnostic to the specific type of rule, and possibly incorporate different kinds of rules simultaneously.

Generalizing constraints across databases. In each method presented in this dissertation, we have discovered constraints on a given, dirty dataset. While various optimizations allow our algorithms to run efficiently on each dataset, the fact still remains that discovery has to be run from scratch, individually on each dataset. If a system is to combine different types of rules, it is clearly problematic to run each discovery algorithm on a large dataset, for every single cleaning session. Moreover, constraints might require validation. One possible avenue to address this situation would be to generalize constraints *across* multiple databases using a machine learning approach. As such, valuable time might be saved when having to discover constraints on a new dirty dataset. The GOLDRUSH system presented in Jarovsky et al. [64] implements a similar idea in order to map fraud detection rules from one dataset to another.

CHAPTER 9

Nederlandse Samenvatting

Doorheen de voorbije decennia, is de hoeveelheid aan data die door computers wordt verwerkt omhoog geschoten tot astronomische proporties. Factoren zoals goedkopere opslag en toegenomen connectiviteit hebben bijgedragen tot deze stortvloed aan data. Terwijl data wordt gegenereerd, bij elkaar geschraapt, en geïntegreerd tegen nooit eerder geziene snelheden, blijkt de kwaliteitscontrole op deze data niet in staat om bij te blijven. Veel van deze data komt voort uit onbetrouwbare bronnen, zoals mogelijk gebrekkige sensoren en overwerkte mensen. Bijgevolg wordt deze enorme massa aan data steeds meer *dirty*.

De onbetrouwbaarheid van data is problematisch voor iedere grote organizatie. Dirty data kost de economie van de VS naar schatting honderden millioenen tot miljarden dollars op jaarbasis [61, 42, 62]. Behalve bedrijven die financiële verliezen lijden, heeft de dirtiness van data ook een serieuze impact op gebieden zoals data analyse, kennisextractie uit databases, en machine learning. Dergelijke applicaties steunen traditioneel op grote hoeveelheden data, en indien de data gewoonlijk dirty is, kan dit leiden tot foute conclusies, gebrekkige modellen, of valse patronen. Zoals het gezegde gaat, “garbage in equals garbage out”.

Het is duidelijk dat datakwaliteit een groot probleem is geworden in datamanagement. Het hoge volume aan data maakt het onmogelijk voor iemand om manueel zijn of haar data op te schonen, en bijgevolg is er een grote vraag naar effectieve methoden om dirty data te corrigeren. Er bestaan verscheidene soorten dirtiness, zoals incomplete data, gedupliceerde data, verouderde data, en inconsistente data. In dit proefschrift ligt de focus op deze laatste categorie: data waarin combinaties van waarden een overtreding vormen van bepaalde logische regels die de data zou moeten volgen. Zulke regels noemen we *kwaliteitsregels*.

Doorheen dit proefschrift hebben we vanuit verschillende invalshoeken het probleem bekeken om deze kwaliteitsregels te vinden, binnen dit paradigma van datakwaliteit op basis van kwaliteitsregels. Het voornaamste probleem hierbij is dat, in een typische situatie, de correcte regels niet gekend zijn. We benaderen dit probleem in dit proefschrift door een gebruiker in te schakelen om kwaliteitsregels te valideren, en door een geschikte interessemaat te gebruiken voor het vinden van regels. De gebruikte technieken zijn veelal geworteld in het gebied van pattern mining, een deelgebied van kennisextractie uit databases, dat vooral is gericht op het ontdekken van interessante associaties tussen zaken of gebeurtenissen.

Overzicht van het Proefschrift

Dit proefschrift bestaat uit zeven hoofdstukken. Hoofdstuk 1 bevat de inleiding, gevolgd door een overzicht van notaties, basisconcepten en gebruikte datasets in hoofdstuk 2. In hoofdstuk 3 hebben we het belangrijkste gerelateerde werk besproken binnen het algemene gebied van datakwaliteit op basis van kwaliteitsregels, en in hoofdstuk 8 vatten we het geleverde werk samen, in combinatie met een conclusie en een perspectief op toekomstmogelijkheden binnen het gebied van kwaliteitsregels voor datakwaliteit. De concrete bijdragen bevinden zich in hoofdstukken 4, 5, 6, en 7, en kunnen als volgt worden samengevat:

- In **Hoofdstuk 4** hebben we bestaande technieken voor het ontdekken van *conditional functional dependencies (CFDs)* herbekeken en veralgemeend. Deze CFDs worden veelvuldig gebruikt in data cleaning, omdat ze eenvoudig te begrijpen zijn, en meer expressieve kracht hebben dan standaard functional dependencies (FDs) of associatieregels. We hebben het ontdekken van CFDs herbeschouwd als een expliciete combinatie van FD ontdekking en itemset mining, en drie verschillende manieren geïdentificeerd waarop deze twee technieken met elkaar gecombineerd kunnen worden. Bijgevolg hebben we aangetoond hoe CFD ontdekking kan profiteren van vooruitgang in beide disciplines. Daarnaast hebben we de de facto pruning strategie voor redundante CFDs veralgemeend, zodat deze ook gebruikt kunnen worden in combinatie met andere zoekstrategieën dan de traditionele breadth-first strategie. In de experimentele sectie hebben we aangetoond dat onze nieuwe methoden, en een correcte keuze van zoekstrategie, een aanzienlijke verbetering in performantie teweeg kan brengen, in vergelijking met het state of the art algoritme CTane voor het vinden van CFDs.
- In **Hoofdstuk 5** hebben we het probleem beschouwd om CFDs te ontdekken die *geldig zijn en nuttig om te corrigeren*, gebaseerd op interactie met een gebruiker. In plaats van deze gebruiker in te schakelen om mogelijke kwaliteitsregels te (in)valideren, hebben we een methode gepresenteerd die enkel gebruik maakt van positieve feedback: de gebruiker corrigeert handmatig een klein deel van de data, en ons algoritme ontdekt vervolgens een CFD die deze partiële correctie *verklaart*. We hebben een scorefunctie geïntroduceerd voor zulke

verklaringen, die aan de grondslag ligt van een efficient “on-demand” algoritme, XPLODE genaamd, dat de beste verklaring vindt voor een gegeven verzameling modificaties. Onze experimenten toonden aan dat onze methode de correcte CFD vindt om data te corrigeren op basis van slechts een klein aantal modificaties, wat leidt tot een noemenswaardige vermindering van de inspanning die van de gebruiker verwacht wordt, in vergelijking met basislijnen zoals confidence. Bovendien is XPLODE robuust tegenover ruis in de modificaties, d.w.z, foutjes die de gebruiker maakt, en beduidend sneller dan een naïeve methode op basis van post-processing, waarbij eerst alle verklaringen worden gevonden en achteraf degene met de hoogste score wordt bepaald.

- In **Hoofdstuk 6** hebben we een *dynamische notie van data kwaliteit* geïntroduceerd, waarbij we een gegeven database als clean beschouwen indien een ontdekkingsalgoritme voor kwaliteitsregels geen overtredingen van deze regels vindt. We hebben deze notie eerst in zijn volledige algemeenheid verwoord, alvorens het probleem concreet te bestuderen voor een eenvoudig te begrijpen taal van kwaliteitsregels, die we *forbidden itemsets* hebben gedoopt. Deze itemsets worden gekenmerkt door het feit dat ze een *lage lift* hebben, en we hebben uitgebreid gemotiveerd waarom zulke itemsets geschikt zijn voor het ontdekken van foutieve waarden in data. Om forbidden itemsets te ontdekken, hebben we een algoritme geïntroduceerd genaamd FBIMiner, en eigenschappen van de lift-maat afgeleid die sterke pruning toelaten. In de experimenten hebben we aangetoond dat forbidden itemsets met een hoge precisie in staat zijn om foutieve waarden te vinden, en dat FBIMiner een goede runtime verwezenlijkt.

- In **Hoofdstuk 7** hebben we een methode voorgelegd voor het *corrigeren* van data in de context van forbidden itemsets, gegeven onze dynamische notie van datakwaliteit. Om te vermijden dat we forbidden itemsets telkens opnieuw moeten berekenen voor iedere kandidaat correctie, introduceren we *almost forbidden itemsets*, wat itemsets zijn die forbidden kunnen worden nadat een zeker aantal correcties zijn toegepast door een correctie-algoritme. Om de runtime nog verder te verbeteren, hebben we voorgesteld om correcties uit te voeren in groepen van een geschikte grootte. Het correctie-algoritme zelf past de dirty objecten aan door suggesties te nemen van de “cleane” gedeelten van de data, die samen worden gegroepeerd, waarna de mate van gelijkenis wordt bepaald op een flexibele wijze, gebruik makend van linkage schema’s die in hiërarchische clustering worden gebruikt. We bewijzen formeel dat de volledige correctiemethode klopt. In de experimentele sectie hebben we de runtime geëvalueerd van het vinden van almost forbidden itemsets. We hebben verder aangetoond dat de runtime van het correctie-algoritme verbeterd kan worden door correcties in parallel uit te voeren, en dat onze correcties de dirty data dichter brengen bij de grondwaarheid.

APPENDIX A

Appendix to Chapter 4

A.1 Search Strategy

We have compared the traditional breadth-first approach to CFD discovery with a depth-first version of the three algorithms. We show the obtained runtimes, in function of minimum support, in Figures A.1- A.4. The experiments were run with a maximum antecedent size of 6. For the Itemset-first and FD-First methodologies, we denote the strategy for the first level in capitals and the strategy for the second level in lowercase, e.g., BFSdfs on FD-First stands for a breadth-first FD mining step and a depth-first itemset mining step.

For the Integrated method, the depth-first version is more efficient in all cases, especially for lower support thresholds. The differences are smaller for Itemset-First, but the combination breadth-first itemset mining and depth-first FD mining generally performs slightly better than the others. In the FD-first case, the different search strategies seem to have very little influence, but the depth-first strategy for both steps is typically marginally faster than the others. This leads to the implementation choices discussed at the beginning of the experimental section.

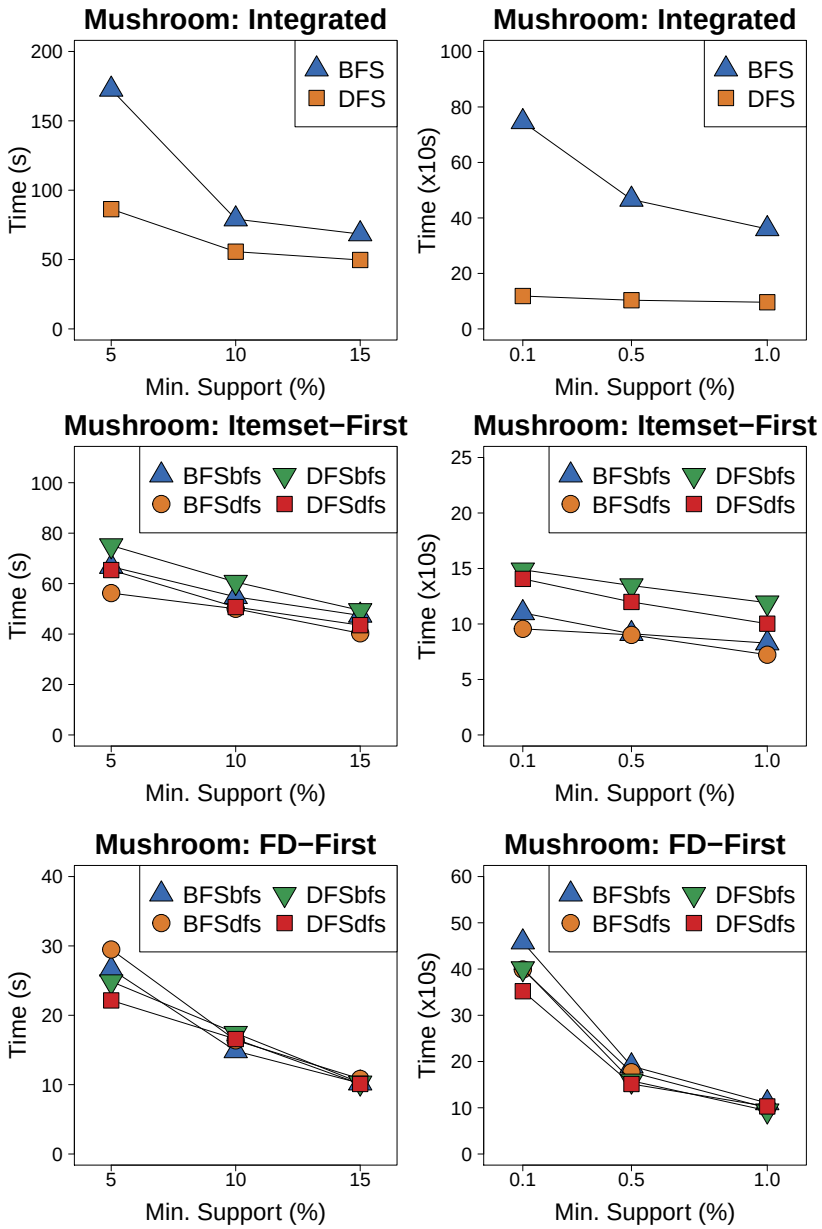


Figure A.1: Comparison of different search strategies for each of the three methodologies (Mushroom).

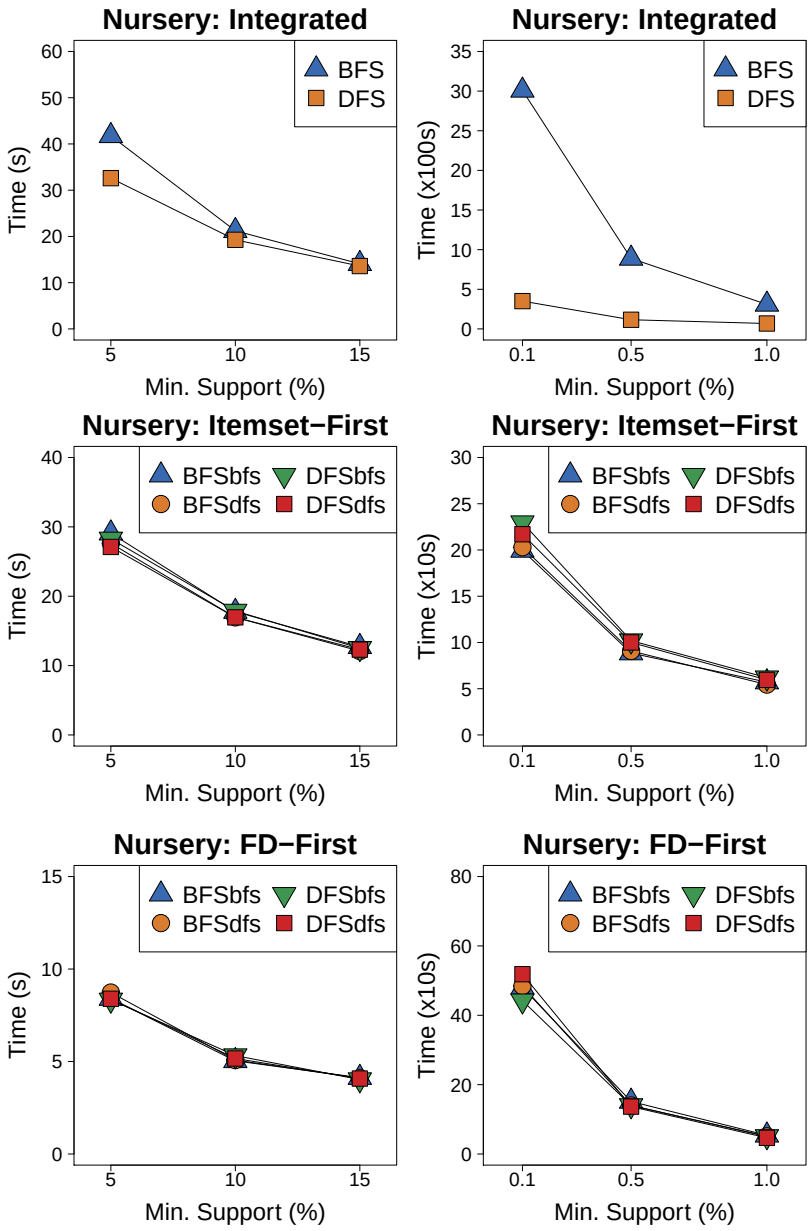


Figure A.2: Comparison of different search strategies for each of the three methodologies (Nursery).

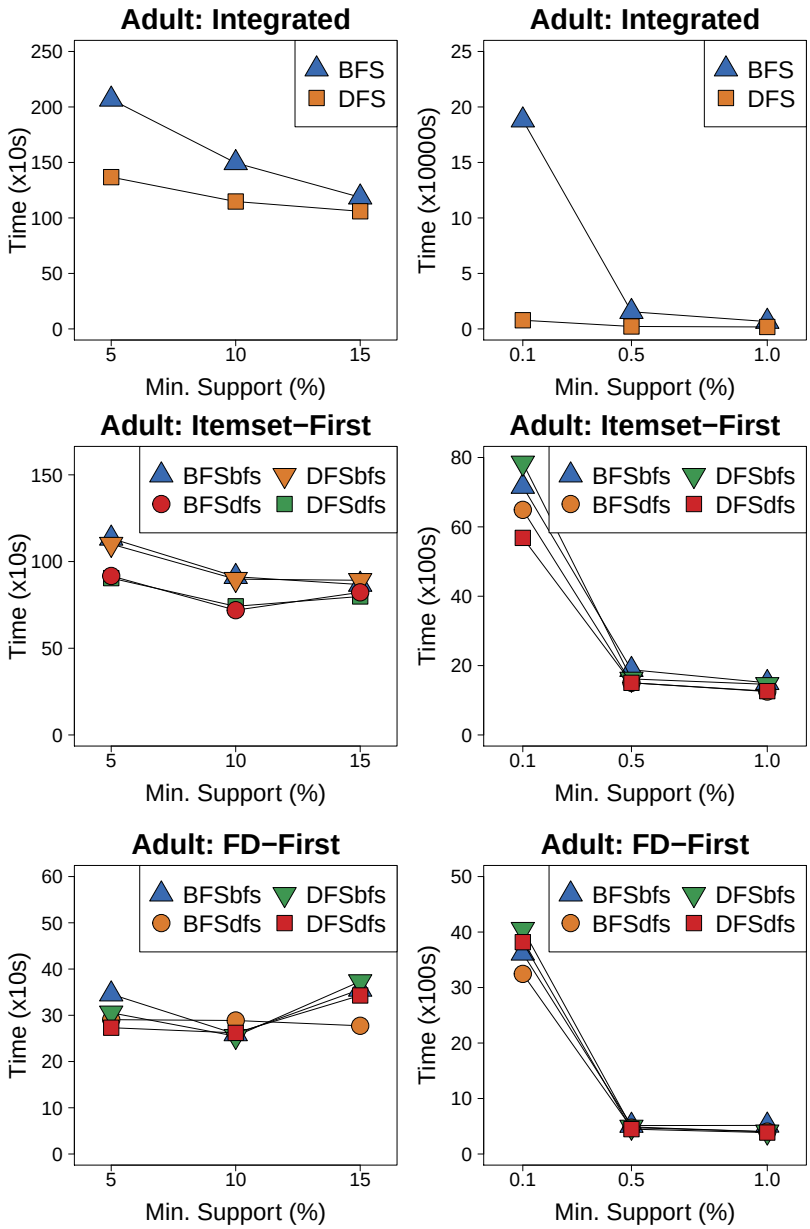


Figure A.3: Comparison of different search strategies for each of the three methodologies (Adult).

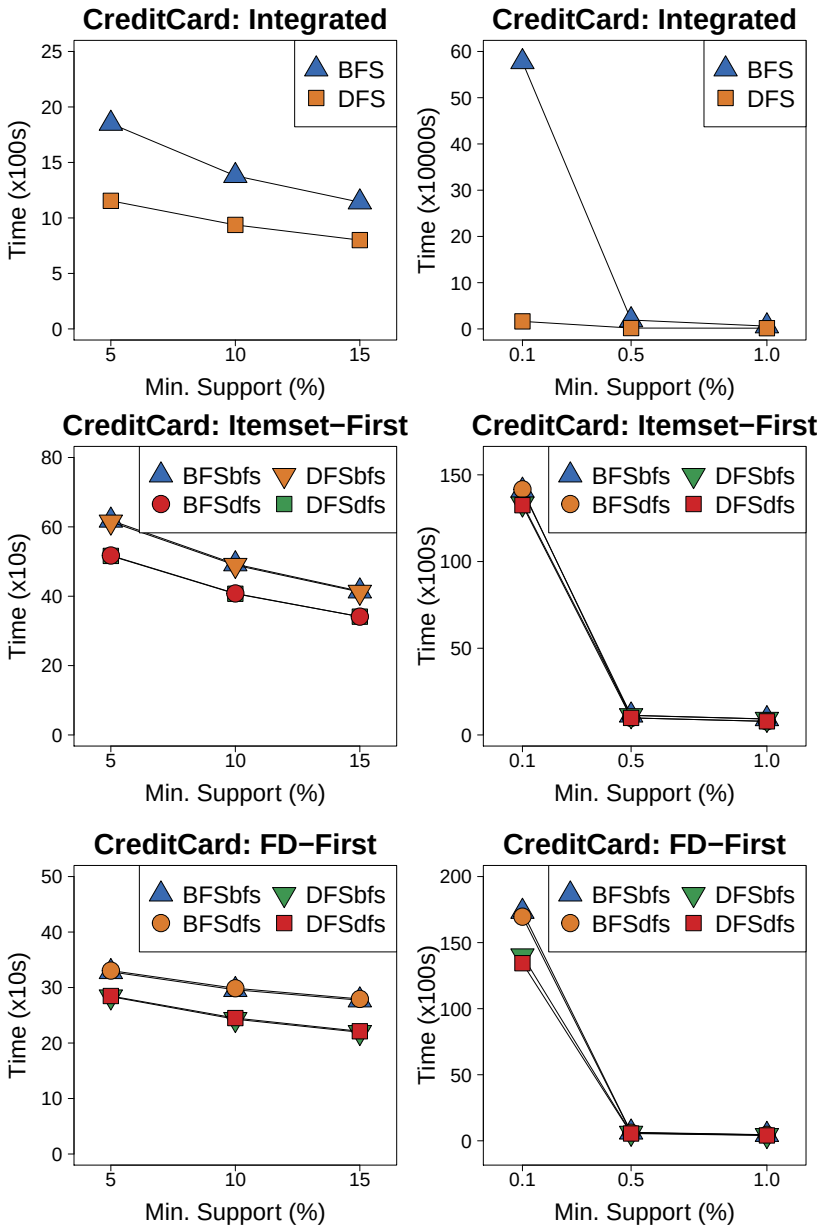


Figure A.4: Comparison of different search strategies for each of the three methodologies (CreditCard).

APPENDIX B

Appendix to Chapter 5

B.1 Proof of Proposition 4

We start by showing that Σ_φ locally explains a Σ_φ-valid set of modifications M if and only if Σ_φ locally explains each $\mathfrak{m} \in M$. Clearly, if Σ_φ locally explains M, then, by definition, Σ_φ is locally explain $\mathfrak{m}$ for every $\mathfrak{m} \in M$. Suppose next that Σ_φ locally explains each $\mathfrak{m} \in M$. We need to show that Σ_φ is an M'-repair explanation for every subset M' of M. We start by showing some useful properties. First, by the definition of $\sigma^{\mathsf{tid}}_{M'}(\cdot)$, we have

$$\sigma^{\mathsf{tid}}_{M'}(D_{dirty}) = \bigcup_{\mathfrak{m} \in M'} \sigma^{\mathsf{tid}}_{\mathfrak{m}}(D_{dirty}), \tag{B.1}$$

Similarly, by the definition of $\sigma_{M'}(\cdot)$ and the validity of M', we have that

$$\sigma_{M'}(D_{dirty} \oplus M') = \bigcup_{\mathfrak{m} \in M'} \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}). \tag{B.2}$$

Furthermore, for a constant CFD $\varphi_{\mathsf{eq}} \in \Sigma_\varphi$, $\mathsf{VIO}(\varphi_{\mathsf{eq}}, D_{dirty} \oplus M')$ consists of all tids corresponding to tuples t in $D_{dirty} \oplus M'$ such that $t \not\models \varphi_{\mathsf{eq}}$. Since M' contains at most one modification per tuple, this implies that either t is unaffected by the modifications in M', or t is a tuple in $D_{dirty} \oplus \mathfrak{m}$ for a (unique) modification $\mathfrak{m} \in M'$. Hence, in both cases, $t \in \mathsf{VIO}(\varphi_{\mathsf{eq}}, D_{dirty} \oplus \mathfrak{m})$ for some $\mathfrak{m} \in M'$. In other words,

$$\mathsf{VIO}(\varphi_{\mathsf{eq}}, D_{dirty} \oplus M') \subseteq \bigcup_{\mathfrak{m} \in M'} \mathsf{VIO}(\varphi_{\mathsf{eq}}, D_{dirty} \oplus \mathfrak{m}).$$

From this, we can infer that

$$\begin{aligned}\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus M') &= \bigcup_{\varphi_{\mathsf{eq}} \in \Sigma_\varphi} \mathsf{VIO}(\varphi_{\mathsf{eq}}, D_{dirty} \oplus M') \\ &\subseteq \bigcup_{\varphi_{\mathsf{eq}} \in \Sigma_\varphi, \mathfrak{m} \in M'} \mathsf{VIO}(\varphi_{\mathsf{eq}}, D_{dirty} \oplus \mathfrak{m}) \\ &= \bigcup_{\mathfrak{m} \in M'} \mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}), \end{aligned} \tag{B.3}$$

where the first equality follows from the definition of $\mathsf{VIO}(\cdot,\cdot)$, the second from our previous observation, and the third equality follows again from the definition of $\mathsf{VIO}(\cdot,\cdot)$. We also note that

$$\begin{aligned}|\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus M')| &= |\mathsf{supp}(\Sigma_\varphi, D_{dirty})| \\ &+ \sum_{\mathfrak{m} \in M'} \big(\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) - \mathsf{supp}(\Sigma_\varphi, D_{dirty})\big), \end{aligned} \tag{B.4}$$

where each $\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) - \mathsf{supp}(\Sigma_\varphi, D_{dirty})$ is simply -1 if the modification reduces the support of Σ_φ, and 0 otherwise. In a similar fashion, we have that:

$$\begin{aligned}|\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus M')| &= |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| \\ &+ \sum_{\mathfrak{m} \in M'} \big(|\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})| - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})|\big). \end{aligned} \tag{B.5}$$

Both Equations (B.4) and (B.5) follow again from the fact that Σ_φ consists of constant CFDs and M' contains at most one modification per tuple.

Given Equations (B.1), (B.2), (B.3), (B.4) and (B.5), we next verify the three conditions of Def. 16 for M'.

▷ **Condition (1)**. We need to verify that $\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty}) < \mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus M')$. Let σ and ν denote $\mathsf{supp}(\Sigma_\varphi, D_{dirty})$ and $|\mathsf{VIO}(\Sigma_\varphi, D_{dirty})|$, respectively; and let $\sigma_\mathfrak{m}$ and $\nu_\mathfrak{m}$ denote $\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})$ and $|\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})|$, respectively. From Equations B.4 and B.5, we can infer that the confidence $\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus M')$ is equal to

$$\frac{\sigma - \nu + \sum_{\mathfrak{m} \in M'} \Big(\big(\sigma_\mathfrak{m} - \nu_\mathfrak{m}\big) - \big(\sigma + \nu\big) \Big)}{\sigma + \sum_{\mathfrak{m} \in M'} \big(\sigma_\mathfrak{m} - \sigma\big)}.$$

By assumption, $\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty}) < \mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})$ for each $\mathfrak{m} \in M'$. Assume that $\mathfrak{m} = (\mathsf{tid}, \mathsf{B}, v_d, v_c)$ and let $s = D_{dirty}[\mathsf{tid}]$ and $t = (D_{dirty} \oplus \mathfrak{m})[\mathsf{tid}]$.

Note that it is not possible that $s \models \Sigma_\varphi$ because by assumption, $\mathsf{VIO}(\Sigma_\varphi, D_{dirty}) \cap \sigma_\mathfrak{m}(D_{dirty})$ is non-empty and the only tuple corresponding to $\sigma_\mathfrak{m}(D_{dirty})$ is s. We thus have that $s \not\models \Sigma_\varphi$, which implies that there exists a $\varphi_{\mathsf{eq}} \in \Sigma_\varphi$ such that $s[\mathsf{X}] = c_{\mathsf{eq}}$. Due to the fact that the CFDs in Σ_φ have mutually exclusive patterns c_{eq}, φ_{eq} is unique. We next distinguish two cases, depending on $\mathfrak{m}$:

(a) $t[\mathsf{X}] = c_{\mathsf{eq}}$. In this case, for the confidence of Σ_φ to increase after modification $\mathfrak{m}$ is applied, the modification must be $\mathfrak{m} = (\mathsf{tid}, \mathsf{A}, s[\mathsf{A}], a_{\mathsf{eq}})$. This modification does not affect the support of φ, brings down the number of violations of Σ_φ by one, and also does not affect supports of any other CFD in Σ_φ (due to mutual exclusive c_{eq} patterns). In other words, in this case

$$\sigma_{\mathfrak{m}} = \sigma$$
$$\nu_{\mathfrak{m}} = \nu - 1.$$

(b) $t[\mathsf{X}] \neq c_{\mathsf{eq}}$. In this case, $t[\mathsf{X}]$ and $s[\mathsf{X}]$ can only differ in a single attribute, say $\mathsf{B} \in \mathsf{X}$. Hence, $\mathfrak{m}$ necessarily must be of the form $(\mathsf{tid}, \mathsf{B}, c_{\mathsf{eq}}[\mathsf{B}], t[\mathsf{B}])$. Furthermore, by assuming Σ_φ-validity the set M' of modifications, we again have two distinct options:

(b1) $t[\mathsf{X}] = c_{\mathsf{eq}'}$ for some $\mathsf{eq}' \neq \mathsf{eq}$. Since Σ_φ is not violated, i.e., the set $\mathsf{VIO}(\Sigma, \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}))$ is empty, it must hold that $t[\mathsf{A}] = a_{\mathsf{eq}'}$. In other words, tuple t no longer violated Σ_φ, and hence the number of violations is reduced by one. Moreover, the support of c_{eq} is reduced by one while the support of $c_{\mathsf{eq}'}$ increases by one, meaning that the support of Σ_φ remains identical. We thus have that:

$$\sigma_{\mathfrak{m}} = \sigma$$
$$\nu_{\mathfrak{m}} = \nu - 1.$$

(b2) There exist no $c_{\mathsf{eq}'}$ in Σ_φ such that $t[\mathsf{X}] = c_{\mathsf{eq}'}$. By definition of Σ_φ-validity, this is only allowed if $t[\mathsf{X}] \not\asymp t_p$, where t_p is the pattern of the CFD φ. In other words, the CFD no longer applies to the tuple. Clearly, this reduces both the support of Σ_φ and the number of violations by one:

$$\sigma_{\mathfrak{m}} = \sigma - 1$$
$$\nu_{\mathfrak{m}} = \nu - 1.$$

In each of the cases (a), (b1) and (b2), the number of violations decreases with 1. Hence,

$$\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus M') = \mathsf{VIO}(\Sigma_\varphi, D_{dirty}) - |M'|.$$

Let $\Delta(\sigma_{M'})$ denote the reduction in support of Σ_φ, i.e., the number of modifications belonging to case (b2), with $\Delta(\sigma_{M'}) \leqslant |M'|$. We can then write the confidence $\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus M')$ as:

$$\frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| + |M'| - \Delta(\sigma_{M'})}{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - \Delta(\sigma_{M'})} \tag{B.6}$$

For $\Delta(\sigma_{M'}) = 0$, expression B.6 is equal to

$$\frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| + |M'|}{\mathsf{supp}(\Sigma_\varphi, D_{dirty})},$$

and for any $\Delta(\sigma_{M'}) > 0$, expression B.6 is greater than or equal to

$$\frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})|}{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - \Delta(\sigma_{M'})},$$

both of which are obviously strictly greater than

$$\mathsf{conf}_{\mathsf{FD}}(\Sigma, D_{dirty}) = \frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})|}{\mathsf{supp}(\Sigma_\varphi, D_{dirty})}$$

for any M' consisting of at least one element. It follows that condition (1) is satisfied.

▷ **Condition (2)**. We need to verify that

$$\mathsf{VIO}(\Sigma_\varphi, D_{dirty}) \cap \sigma^{\mathsf{tid}}_{M'}(D_{dirty})$$

is non-empty. From Equation B.1, this is equivalent to requiring

$$\bigcup_{\mathfrak{m} \in M'} \mathsf{VIO}(\Sigma_\varphi, D_{dirty}) \cap \sigma^{\mathsf{tid}}_{\mathfrak{m}}(D_{dirty})$$

to be non-empty. This follows from the fact that Σ_φ satisfied condition (2) for every $\mathfrak{m} \in M'$.

▷ **Condition (3)**. We need to verify that

$$\mathsf{VIO}(\Sigma_\varphi, \sigma_{M'}(D_{dirty} \oplus M'))$$

is empty. Note that for constant CFDs this is equivalent to checking whether

$$\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus M') \cap \sigma_{M'}(D_{dirty} \oplus M')$$

is empty. Using Equations B.1 and B.3, we know that this set is included in

$$\bigcup_{\mathfrak{m} \in M'} (\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) \cap \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}),$$

which is known to be empty, due the fact that Σ_φ satisfies condition (3) for each $\mathfrak{m} \in M'$. It follows that

$$\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus M') \cap \sigma_{M'}(D_{dirty} \oplus M')$$

is empty, and hence condition (3) is satisfied.

B.2 Proof of Proposition 5

We prove this by showing for any set of modifications $M \subseteq \mathfrak{M}$, if Σ_φ is an M-repair explanation then φ is also an M-repair explanation.

▷ **Condition 2:** $\mathsf{VIO}(\varphi, D_{dirty}) \cap \sigma^{\mathsf{tid}}_{M}(D_{dirty})$ is not empty:
This follows from the fact that any violation of Σ_φ, is also a violation of φ. Indeed, let $\varphi_{\mathsf{eq}} = (\mathsf{X} \to \mathsf{A}, (c_{\mathsf{eq}}, a_{\mathsf{eq}}))$ be a constant CFD violated in a tuple $t \in$

D_{dirty}. Then $t[\mathsf{X}] = c_{\mathsf{eq}}$ and $t[\mathsf{A}] \neq a_{\mathsf{eq}}$. By construction of Σ_φ, it must hold that there exists a tuple $t' \in D_{dirty}$ such that $t'[\mathsf{X}] = c_{\mathsf{eq}}$ and $t'[\mathsf{A}] = a_{\mathsf{eq}}$. Clearly, t and t' together violate the CFD φ. It follows that $\mathsf{VIO}(\Sigma_\varphi, D_{dirty}) \cap \sigma_M^{\mathsf{tid}}(D_{dirty})$ is included in $\mathsf{VIO}(\varphi, D_{dirty}) \cap \sigma_M^{\mathsf{tid}}(D_{dirty})$. Since the former is non-empty, so is the latter.

▷ **Condition 3:** $\mathsf{VIO}(\varphi, \sigma_M(D_{dirty} \oplus M))$ is empty
We show this by contradiction. Assume there exist two tuples $t, t' \in \sigma_M(D_{dirty} \oplus M)$ that violate $\varphi : (X \to A, (t_p, \_))$, i.e., $t[\mathsf{X}] = t'[\mathsf{X}]$ and $t[\mathsf{A}] \neq t'[\mathsf{A}]$. By construction Σ_φ, there must exist some $\varphi_{\mathsf{eq}} : (X \to A, (c_{\mathsf{eq}}, a_{\mathsf{eq}}))$ such that $c_{\mathsf{eq}} = t[\mathsf{X}] = t'[\mathsf{X}]$. Since φ_{eq} is not violated in $\sigma_M(D_{dirty} \oplus M)$, it must therefore also hold that $t[\mathsf{A}] = t'[\mathsf{A}] = a_{\mathsf{eq}}$, contradicting the assumption that $t[\mathsf{A}] \neq t'[\mathsf{A}]$, and hence no such t and t' violating φ can exist in $\sigma_M(D_{dirty} \oplus M)$. It follows that $\mathsf{VIO}(\Sigma_\varphi, \sigma_M(D_{dirty} \oplus M))$ is empty implies that $\mathsf{VIO}(\varphi, \sigma_M(D_{dirty} \oplus M))$ is empty.

▷ **Condition 1:** $\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty} \oplus M) > \mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty})$
In the proof of Proposition 4 (expression B.6), it was shown that the confidence Σ_φ in $D_{dirty} \oplus M$ is equal to:

$$\frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| + |M| - \Delta(\sigma_M)}{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - \Delta(\sigma_M)}$$

Where $\Delta(\sigma_M)$ denotes the reduction in the support of Σ_φ as a result of the modifications M, with $\Delta(\sigma_{M'}) \leqslant |M|$.

Recall that the confidence of a CFD is defined based on a set D'_{dirty}, containing the minimum amount of tuples that need to be altered or removed from D_{dirty} for φ to be satisfied. Since the union of constants is constructed based on the most frequent value in each equivalence class, $|D'_{dirty}|$ is equal to $|\mathsf{VIO}(\Sigma_\varphi, D_{dirty})|$. Indeed, such a minimal set D'_{dirty} contains exactly those tuples that violate the constant CFDs in Σ_φ. We thus have that:

$$\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty}) = \frac{\mathsf{supp}(\varphi, D_{dirty}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})|}{\mathsf{supp}(\varphi, D_{dirty})}.$$

Moreover, since every modification $\mathfrak{m}$ that is locally explained by Σ_φ must occur in a tuple violated by Σ_φ, it follows that every such modification pertains to the set D'_{dirty}. Clearly, when the tuple is modified to satisfy some $c_{\mathsf{eq}} \in \Sigma_\varphi$, it now belongs to the most frequent value in its equivalence class, and is no longer a part of D'_{dirty}. It follows that $|(D_{dirty} \oplus M)'| = |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| - |M|$.

The support of the CFD Σ_φ is reduced in only one case, namely when the modification is such that t_p no longer applies, i.e., $t[\mathsf{X}] \not\asymp t_p$, where t is the modified tuple. It is clear that such a modification also reduces the support of φ. Hence, $\mathsf{supp}(\varphi, D_{dirty} \oplus M) = \mathsf{supp}(\varphi, D_{dirty}) - \Delta(\sigma_M)$.

We thus have that:

$$\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty} \oplus M) = \frac{\mathsf{supp}(\varphi, D_{dirty}) - \Delta(\sigma_M) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| + |M|}{\mathsf{supp}(\varphi, D_{dirty}) - \Delta(\sigma_M)}.$$

Analogous to the proof of Proposition 4, it follows that

$$\mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty} \oplus M) > \mathsf{conf}_{\mathsf{FD}}(\varphi, D_{dirty}). \quad \square$$

B.3 Proof of Proposition 6

Let $\mathfrak{m} = (\mathsf{tid}, \mathsf{B}, v_d, v_c) \in \mathfrak{M}$, let $s = D_{dirty}[\mathsf{tid}]$ and $t = (D_{dirty} \oplus \mathfrak{m})[\mathsf{tid}]$. Let Σ_φ be the union of constant CFDs constructed from $\varphi = (\mathsf{X} \to \mathsf{A}, (t_p, \_))$. We only deal with the variable CFD case here, the constant CFD case is similar (and in fact easier).

$\boxed{\Rightarrow}$ We first show that the conditions in Proposition 6 are sufficient for Σ_φ to locally explain $\mathfrak{m}$. To this aim we verify that conditions (1), (2) and (3) in Definition 16 are satisfied.

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 $s[\mathsf{A}] \neq a_{\mathsf{eq}}$, then clearly Σ_φ is violated on s, and $\mathsf{VIO}(\Sigma_\varphi, D_{dirty}) \cap \sigma^{\mathsf{tid}}_{\mathfrak{m}}(D_{dirty})$ is not empty, satisfying condition (2).

We now verify that conditions (1) and (3) are satisfied for each of the 3 cases in which Σ_φ is defined to explain $\mathfrak{m}$:

Case 1. Assume $t[\mathsf{A}] = a_{\mathsf{eq}}$, i.e., $\mathfrak{m}$ changes s into the tuple t such that $t \models \varphi_{\mathsf{eq}}$. Since φ_{eq} is the only CFD in Σ_φ that applies to tuple t, by construction of Σ_φ, it follows that Σ_φ is satisfied in the modified tuple, and hence $\mathsf{VIO}(\varphi, \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}))$ is empty, satisfying condition (3) in Definition 16. Moreover, a violation has been resolved, while the support of Σ_φ remains unchanged. Indeed, s supported c_{eq} in D_{dirty} and t remains to do so in $D_{dirty} \oplus \mathfrak{m}$. We thus have that

$$\begin{aligned}
&\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) \\
&= \frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})|}{\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})} \\
&= \frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - (|\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| - 1)}{\mathsf{supp}(\Sigma_\varphi, D_{dirty})}.
\end{aligned}$$

Hence, also condition (1) in Definition 16, namely

$$\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) > \mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty})$$

is satisfied.

Case 2. Assume that there exists another $\varphi_{\mathsf{eq}'} \in \Sigma_\varphi$ such that $t[\mathsf{X}] = c_{\mathsf{eq}'}$ and $t[\mathsf{A}] = a_{\mathsf{eq}'}$, i.e., t satisfies some other CFD in Σ_φ. Again, $\varphi_{\mathsf{eq}'}$ is by definition the only CFD in Σ_φ that applies to tuple t, implying that Σ_φ is satisfied in the modified tuple. Hence, $\mathsf{VIO}(\varphi, \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}))$ is empty, satisfying condition (3). We have that

$$\mathsf{supp}(\varphi_{\mathsf{eq}}, D_{dirty} \oplus \mathfrak{m}) = \mathsf{supp}(\varphi_{\mathsf{eq}}, D_{dirty}) - 1,$$

and

$$\mathsf{supp}(\varphi_{\mathsf{eq}'}, D_{dirty} \oplus \mathfrak{m}) = \mathsf{supp}(\varphi_{\mathsf{eq}}, D_{dirty}) + 1,$$

meaning that the support of Σ_φ again remains unchanged. Since one violation has been resolved, the analysis of confidence is identical to the previous case, and hence condition (1) in Definition 16 is also satisfied.

Case 3. Finally, assume that $t[\mathsf{X}] \not\asymp t_p$, i.e., φ no longer applies to t. It trivially follows that $\mathsf{VIO}(\varphi, \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}))$ is empty, satisfying condition (3). The modification has resolved a violation, while the support of Σ_φ has been reduced by one. We thus have that:

$$\begin{aligned}
&\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) \\
&= \frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) - |\mathsf{VIO}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})|}{\mathsf{supp}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m})} \\
&= \frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) + 1 - (|\mathsf{VIO}(\Sigma_\varphi, D_{dirty})| - 1)}{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) + 1} \\
&= \frac{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) - (|\mathsf{VIO}(\Sigma_\varphi, D_{dirty})|) + 2}{\mathsf{supp}(\Sigma_\varphi, D_{dirty}) + 1}.
\end{aligned}$$

And hence, it holds that

$$\mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty} \oplus \mathfrak{m}) > \mathsf{conf}_{\mathsf{FD}}(\Sigma_\varphi, D_{dirty}).$$

From the above, we conclude that if the conditions in Proposition 6 are satisfied, then the three conditions in Definition 16 are satisfied, and thus Σ_φ is a $\mathfrak{m}$-repair explanation, or equivalently for single modifications, Σ_φ locally explains the modification $\mathfrak{m}$.

$\boxed{\Leftarrow}$ We next verify that the conditions in Proposition 6 are also necessary for Σ_φ to explain a modification $\mathfrak{m}$.

If there does not exist 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 $s[\mathsf{A}] \neq a_{\mathsf{eq}}$, then by construction of Σ_φ, the tuple s does not violate any CFD in Σ_φ. It follows that $\mathsf{VIO}(\Sigma_\varphi, D_{dirty}) \cap \sigma^{\mathsf{tid}}_{\mathfrak{m}}(D_{dirty})$ is empty, and hence condition (2) is not satisfied. Therefore, Σ_φ cannot locally explain $\mathfrak{m}$.

Alternatively, assume that there does exist 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 $s[\mathsf{A}] \neq a_{\mathsf{eq}}$. We show that, if none of the three cases in Proposition 6 hold in which Σ_φ is defined to explain $\mathfrak{m}$, then condition (3) in Definition 16 cannot be satisfied. We distinguish between two cases, depending on the attribute B which was modified:

- Assume $\mathsf{B} = \mathsf{A}$, i.e., the modification relates to the consequent of the CFD φ. In this case, only case (1) is relevant. If this does not hold, then $t[\mathsf{A}] \neq a_{\mathsf{eq}}$. However, no changes were made to the other values in t, meaning that φ_{eq} still applies to t, and is thus still violated. It follows that Σ_φ is violated after the modification, and $\mathsf{VIO}(\varphi, \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}))$ is not empty. Hence, Σ_φ does not locally explain $\mathfrak{m}$.
- Assume $\mathsf{B} \in \mathsf{X}$, i.e., the modification relates to the antecedent of the CFD φ. Both scenarios (2) and (3) are relevant. If neither scenario applies, then $t[\mathsf{X}] \asymp t_p$, and thus the CFD φ still applies. It remains to show that the modified tuple t necessarily violates Σ_φ. Note that, since

$\mathfrak{m}$ is assumed to be Σ-valid, there must exist exactly one CFD $\varphi_{eq'} = (X \to A, (c_{eq'}, a_{eq'})) \in \Sigma_\varphi$ such that $t[X] = c_{eq'}$. However, since scenario (2) does not hold, $t[A] \neq a_{eq'}$. Clearly, t violates $\varphi_{eq'}$, and hence $\mathsf{VIO}(\varphi, \sigma_{\mathfrak{m}}(D_{dirty} \oplus \mathfrak{m}))$ is again not empty, violating condition (3).

From the above, it follows that if the conditions in Proposition 6 do not apply, then Σ_φ does not locally explain $\mathfrak{m}$. This proves the proposition. □

B.4 Number of Global Explanations

In Table B.2, we show the number of candidate global explanations found on the various datasets. These numbers were obtained for 1 modification, 50% of modifications, and all modifications. We see that the numbers are high, showing the need for a scoring function to determine the best explanation. Note that the 0 on SP500 for 1 modification is because the target explanation (and many others) did not meet the support/confidence thresholds.

B.5 Additional Details On Experiments

We show the CFDs used in the experiments in Table B.1.

Dataset	**CFDs**
Abalone	(viscera, whole, rings=10) → height
	(whucked, whole, height, diameter) → rings
	(shell, whole, length, rings=9) → height
Adult	(maritalstatus=Never-married, race=White, sex=Male, relationship=Own-child, age=18-21) → income=LessThan50K
	(maritalstatus=Never-married, age=18-21, education=HS-grad) → income=LessThan50K
	(education=Some-college, relationship=Husband) → sex=Male
Soccer	(season, position, surname, stadium=King Power Stadium) → team
	(city, birthplace, season=2013) → position
	(position, surname, city=Solna) → season
SP500	(g, e, d) → f
	(g, f, b) → d
	(g, f, c) → d

Table B.1: CFDs used in the experiments.

Table B.2: Number of candidate global explanations for various numbers of modifications.

Dataset	CFD	Nr. Modifications	Nr. Global Explanations
Abalone	1	1	144
		41	1237
		83	1141
	2	1	120
		41	1078
		83	1289
	3	1	153
		41	775
		83	774
Adult	1	1	349
		244	2625
		488	1578
	2	1	650
		244	2602
		488	1686
	2	1	455
		244	5753
		488	2571
Soccer	1	1	218
		1000	2542
		2000	594
	2	1	212
		1000	2266
		2000	868
	3	1	337
		1000	2643
		2000	723
SP500	1	1	0
		612	89
		1225	82
	2	1	0
		612	121
		1225	102
	3	1	0
		612	118
		1225	101

APPENDIX C

Appendix to Chapter 7

C.1 Continuation of the Proof of Proposition 13

We present the full case analysis underlying Proposition 13. Let $(\Delta_I, \Delta_J, \Delta_{I\setminus J})$ be such that Δ_I is either -1 (decrement), 0 (no change), or $+1$ (increment), and similarly for Δ_J and $\Delta_{I\setminus J}$. We need to show that for all triples $(\Delta_I, \Delta_J, \Delta_{I\setminus J})$,

$$\frac{|\mathcal{D}| \times (\mathsf{supp}(I,\mathcal{D}) + \Delta_I)}{(\mathsf{supp}(J,\mathcal{D}) + \Delta_J) \times (\mathsf{supp}(I\setminus J,\mathcal{D}) + \Delta_{I\setminus J})}$$

is either larger than

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

which corresponds to the triple $(-1,0,-1)$; or larger than

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

which corresponds to the triple $(0,+1,0)$.

Although there are $3^3 = 27$ different triples $(\Delta_I, \Delta_J, \Delta_{I\setminus J})$ we can immediately rule out certain combinations which are impossible in practice.

Firstly, if $\Delta_I =$"$+1$", then neither Δ_J nor $\Delta_{I\setminus J}$ can be "-1". Clearly, if a single modification reduces the support of J or $I\setminus J$, the modified object no longer contains these itemsets, and hence cannot contain a new instance of their superset I. Moreover, at least one of Δ_J or $\Delta_{I\setminus J}$ must be "$+1$" for the support of I to increase. This rules out 6 cases: $(+1,0,0)$, $(+1,0,-1)$, $(+1,-1,0)$, $(+1,+1,-1)$, $(+1,-1,+1)$ and $(+1,-1,-1)$.

A similar argument can be made when $\Delta_I =$"-1", in which case neither Δ_J nor $\Delta_{I\setminus J}$ can be "$+1$", and at least one of Δ_J or $\Delta_{I\setminus J}$ must be "-1" for the support of I to decrease. This again rules out 6 cases: $(-1,0,0)$, $(-1,0,+1)$, $(-1,+1,0)$, $(-1,-1,+1)$, $(-1,+1,-1)$ and $(-1,+1,+1)$.

Furthermore, if $\Delta_I =$"0", then Δ_J and $\Delta_{I\setminus J}$ cannot both be "$+1$" or "-1". Indeed, if a modification were to introduce (resp. remove) an occurence of both J and $I\setminus J$, then clearly it must also introduce (resp. remove) an occurrence of their union, $J\cup(I\setminus J)=I$, which contradicts the fact that $\Delta_I = 0$". This rules out 2 more cases, $(0,+1,+1)$ and $(0,-1,-1)$.

Finally, the case when $(\Delta_I,\Delta_J,\Delta_{I\setminus J}) = (0,0,0)$ is not very interesting since it does not change the lift, and will always end up being higher than any modification that reduces the lift. We can therefore ignore this in our case analysis.

This leaves us with the following 12 cases:

(1)	$(0,0,+1)$	(2)	$(0,+1,0)$
(3)	$(0,0,-1)$	(4)	$(0,-1,0)$
(5)	$(0,-1,+1)$	(6)	$(0,+1,-1)$
(7)	$(+1,0,+1)$	(8)	$(+1,+1,0)$
(9)	$(+1,+1,+1)$	(10)	$(-1,0,-1)$
(11)	$(-1,-1,1)$	(12)	$(-1,-1,-1)$

We will show that cases (2) and (10) lead to a maximum possible decrease in lift. Furthermore (2) and (10) are incomparable, i.e., depending on the instance and itemsets under consideration, (2) may lead more decrease in lift than (10), or vice versa, after one modification is made to $\mathcal{D}$. In fact, we will show that cases (1)-(12) form two distinct partial orderings. Let $(i) \leqslant (j)$ denote that (i) leads to more (or equal) decrease in lift than (j):

$$\begin{array}{lllll} & \leqslant (1) & \leqslant (9) & \leqslant (8) & \leqslant (7) \\ (2) & \leqslant (6) & \leqslant (5) & & \qquad (\clubsuit) \\ (10) & \leqslant (11) & \leqslant (12) & \leqslant (3) & \leqslant (4) \end{array}$$

We will now prove the inequalities that underlie the orderings of the cases. Recall that we assumed that $\mathsf{supp}(I\setminus J,\mathcal{D}) = \mathsf{supp}(J,\mathcal{D}) + a$ for some constant $a \geqslant 0$.

$\triangleright$ Consider cases (2) and (1). To show that $(2) \leqslant (1)$, we verify that

$$\frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a)} \quad \leqslant \quad \frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+a+1)}$$

holds. It is a straightforward exercise to check that this inequality always holds. Indeed, it boils down to verifying that

$$(\mathsf{supp}(J,\mathcal{D}) + 1) \times (\mathsf{supp}(J,\mathcal{D}) + a) \geqslant \mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D}) + a + 1),$$

which in turn is equivalent to checking that

$$\big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a+1)\mathsf{supp}(J,\mathcal{D}) + a \geqslant \big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a+1)\mathsf{supp}(J,\mathcal{D}).$$

This always holds since $a \geqslant 0$. Hence, it holds that $(2) \leqslant (1)$.
$\triangleright$ Cases (1) and (9): To show that $(1) \leqslant (9)$ we verify that

$$\frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+a+1)} \leqslant \frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})+1)}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a+1)}$$

holds. This is equivalent to checking whether

$$\frac{\mathsf{supp}(I,\mathcal{D})}{\mathsf{supp}(J,\mathcal{D})} \leqslant \frac{\mathsf{supp}(I,\mathcal{D})+1}{\mathsf{supp}(J,\mathcal{D})+1},$$

or

$$\mathsf{supp}(I,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+1) \leqslant \mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(I,\mathcal{D})+1)$$

is true. This holds since $\mathsf{supp}(I,\mathcal{D}) \leqslant \mathsf{supp}(J,\mathcal{D})$.
$\triangleright$ Cases (9) and (8): To show that $(9) \leqslant (8)$ we verify that

$$\frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})+1)}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a+1)} \leqslant \frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})+1)}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a)}$$

holds. This follows immediately from the fact that

$$\frac{1}{\mathsf{supp}(J,\mathcal{D})+a+1} \leqslant \frac{1}{\mathsf{supp}(J,\mathcal{D})+a}.$$

$\triangleright$ Cases (8) and (7): To show that $(9) \leqslant (8)$ we verify that

$$\frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})+1)}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a)} \leqslant \frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})+1)}{\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+a+1)},$$

holds. This pours down to verifying that

$$(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a) \geqslant (\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+a+1),$$

which in turn is equivalent to checking that

$$\big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a+1)\mathsf{supp}(J,\mathcal{D}) + a \geqslant \big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a+1)\mathsf{supp}(J,\mathcal{D})$$

holds. This follows again from the fact that $a \geqslant 0$.
$\triangleright$ Cases (2) and (6): To show that $(2) \leqslant (6)$ we verify that

$$\frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a)} \leqslant \frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a-1)}$$

holds. This is equivalent to checking

$$(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a) \geqslant (\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a-1),$$

or, in other words, whether

$$\big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a+1)\mathsf{supp}(J,\mathcal{D}) + a \geqslant \big(\mathsf{supp}(J,\mathcal{D})\big)^2 + a\,\mathsf{supp}(J,\mathcal{D}) + a - 1,$$

holds. Clearly, $a+1 > a$ and $a > a-1$ and hence this inequality is trivially satisfied.
$\triangleright$ Cases (6) and (5): To show that $(6) \leqslant (5)$ we verify that

$$\frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a-1)} \leqslant \frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{(\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a+1)}$$

holds. This pours down to checking

$$(\mathsf{supp}(J,\mathcal{D})+1) \times (\mathsf{supp}(J,\mathcal{D})+a-1) \geqslant (\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a+1),$$

or equivalently, whether

$$\big(\mathsf{supp}(J,\mathcal{D})\big)^2 + a\,\mathsf{supp}(J,\mathcal{D}) + a - 1 \geqslant \big(\mathsf{supp}(J,\mathcal{D})\big)^2 + a\,\mathsf{supp}(J,\mathcal{D}) - a - 1$$

holds. This inequality is satisfied because $a \geqslant 0$ and hence $a \geqslant -a$.
$\triangleright$ Cases (10) and (11): To show that $(10) \leqslant (11)$ we verify that

$$\frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})-1)}{\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+a-1)} \leqslant \frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})-1)}{(\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a)}$$

holds. This is equivalent to checking

$$\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D}) + a - 1) \geqslant (\mathsf{supp}(J,\mathcal{D}) - 1) \times (\mathsf{supp}(J,\mathcal{D}) + a)$$

or, whether

$$\big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a-1)\mathsf{supp}(J,\mathcal{D}) \geqslant \big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a-1)\mathsf{supp}(J,\mathcal{D}) - a$$

is true. Again, this follows from $a \geqslant 0$ and hence $0 \geqslant -a$.
$\triangleright$ Cases (11) and (12): To show that $(11) \leqslant (12)$ we verify that

$$\frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})-1)}{(\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a)} \leqslant \frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})-1)}{(\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a-1)}$$

holds. We can equivalently check

$$(\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a) \geqslant (\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a-1),$$

or whether

$$\big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a-1)\mathsf{supp}(J,\mathcal{D}) \geqslant \big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a-2)\mathsf{supp}(J,\mathcal{D}) - a + 1$$

is true. This inequality holds since $\mathsf{supp}(J,\mathcal{D}) \geqslant 1$ and $a \geqslant 0$. Indeed, from these we have that $(a-2)\mathsf{supp}(J,\mathcal{D}) - a + 1 \leqslant (a-1)\mathsf{supp}(J,\mathcal{D})$, from which the inequality follows.
$\triangleright$ Cases (12) and (3): To show that $(12) \leqslant (3)$ we verify that

$$\frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D})-1)}{(\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a-1)} \leqslant \frac{|\mathcal{D}'| \times (\mathsf{supp}(I,\mathcal{D}))}{\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+a-1)}$$

holds. It suffices to show that

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

which holds since $\mathsf{supp}(I,\mathcal{D}) \leqslant \mathsf{supp}(J,\mathcal{D})$.
$\triangleright$ Cases (3) and (4): To show that (3) $\leqslant$ (4) we verify that

$$\frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D})+a-1)} \leqslant \frac{|\mathcal{D}'| \times \mathsf{supp}(I,\mathcal{D})}{(\mathsf{supp}(J,\mathcal{D})-1) \times (\mathsf{supp}(J,\mathcal{D})+a)}$$

holds. This is equivalent to checking

$$\mathsf{supp}(J,\mathcal{D}) \times (\mathsf{supp}(J,\mathcal{D}) + a - 1) \geqslant (\mathsf{supp}(J,\mathcal{D}) - 1) \times (\mathsf{supp}(J,\mathcal{D}) + a)$$

or

$$\big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a-1)\mathsf{supp}(J,\mathcal{D}) \geqslant \big(\mathsf{supp}(J,\mathcal{D})\big)^2 + (a-1)\mathsf{supp}(J,\mathcal{D}) - a$$

holds. This follows immediately from $a \geqslant 0$ and hence $0 \geqslant -a$.

The previous case comparisons prove the partial orderings (♣). It remains to show that (2) and (10) are not comparable in general. We show this by providing counter examples. Assume first that (2) $\geqslant$ (10). However, when choosing $\mathsf{supp}(I,\mathcal{D}) = 2$, $\mathsf{supp}(J,\mathcal{D}) = 2$ and $a = 1$, we have that:

$$\frac{|\mathcal{D}'| \times 2}{3 \times 3} < \frac{|\mathcal{D}'| \times 1}{2 \times 2}$$

Which contradicts the assumption. Hence, (2) $\not\geqslant$ (10).

Assume next that (2) $<$ (10). However, when choosing $\mathsf{supp}(I,\mathcal{D}) = 2$, $\mathsf{supp}(J,\mathcal{D}) = 2$ and $a = 3$, we have that:

$$\frac{|\mathcal{D}'| \times 2}{3 \times 5} > \frac{|\mathcal{D}'| \times 1}{2 \times 4}$$

Which again contradicts the assumption. Hence, (2) $\not<$ (10), which proves that (2) and (10) are not comparable. This concludes our proof that (2) and (10) are indeed the two cases that cause a maximum reduction in lift.

C.2 Continuation of the Proof of Proposition 19

We expand on the proof given for Proposition 19, showing that

$$\tau < \frac{|\mathcal{D}| \times \mathsf{supp}(J,\mathcal{D}'_{k-1})}{\mathsf{supp}(S,\mathcal{D}'_{k-1}) \times \mathsf{supp}(J \setminus S,\mathcal{D}'_{k-1})} \Rightarrow$$
$$\tau < \frac{|D| \times \mathsf{supp}(J,\mathcal{D}'_{k-1}) + 1}{(\mathsf{supp}(S,\mathcal{D}'_{k-1}) + 1) \times (\mathsf{supp}(J \setminus S,\mathcal{D}'_{k-1}) + 1)},$$

holds for all $\tau < 3/4$.

The implication is trivially satisfied if

$$\frac{|\mathcal{D}| \times (\mathsf{supp}(J, \mathcal{D}'_{k-1}) + 1)}{(\mathsf{supp}(S, \mathcal{D}'_{k-1}) + 1) \times (\mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1}) + 1)} \geqslant \frac{|D| \times \mathsf{supp}(J, \mathcal{D}'_{k-1})}{\mathsf{supp}(S, \mathcal{D}'_{k-1}) \times \mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1})}.$$

We therefore consider the case when the previous inequality is not satisfied. In other words, we only need to consider the case where the lift drops after a modification. To simplify notation, we let $\sigma_J = \mathsf{supp}(J, \mathcal{D}'_{k-1})$, $\sigma_S = \mathsf{supp}(S, \mathcal{D}'_{k-1})$ and $\sigma_{J\setminus S} = \mathsf{supp}(J \setminus S, \mathcal{D}'_{k-1})$. We therefore consider the case when

$$\frac{|\mathcal{D}|(\sigma_J + 1)}{(\sigma_S + 1) \times (\sigma_{J\setminus S} + 1)} < \frac{|D| \times \sigma_J}{\sigma_S \times \sigma_{J\setminus S}} \tag{C.1}$$

holds. We now proof the proposition by contradiction. That is, suppose that there exists a $\tau < 3/4$ such that $\tau < \frac{|D|\times\sigma_J}{\sigma_S\times\sigma_{J\setminus S}}$ but $\frac{|\mathcal{D}|(\sigma_J+1)}{(\sigma_S+1)\times(\sigma_{J\setminus S}+1)} \leqslant \tau$.

We start by eliminating σ_J from the latter expression. To this aim, observe that (C.1) implies that

$$\frac{|\mathcal{D}|}{\sigma_S + \sigma_{J\setminus S} + 1} < \frac{|D| \times \sigma_J}{\sigma_S \times \sigma_{J\setminus S}},$$

or, equivalently, that

$$\frac{\sigma_S \times \sigma_{J\setminus S}}{\sigma_S + \sigma_{J\setminus S} + 1} < \sigma_J. \tag{C.2}$$

Now, observe the following:

$$\begin{aligned}
\tau &\geqslant \frac{|\mathcal{D}|(\sigma_J + 1)}{(\sigma_S + 1) \times (\sigma_{J\setminus S} + 1)} \\
&> \frac{|\mathcal{D}| \times (\frac{\sigma_S\times\sigma_{J\setminus S}}{\sigma_S+\sigma_{J\setminus S}+1} + 1)}{(\sigma_S + 1) \times (\sigma_{J\setminus S} + 1)} && \text{(Using (C.2))} \\
&= \frac{|\mathcal{D}| \times \frac{\sigma_S\times\sigma_{J\setminus S}+\sigma_S+\sigma_{J\setminus S}+1}{\sigma_S+\sigma_{J\setminus S}+1}}{(\sigma_S + 1) \times (\sigma_{J\setminus S} + 1)} \\
&= \frac{|\mathcal{D}| \times \frac{(\sigma_S+1)\times(\sigma_{J\setminus S}+1)}{\sigma_S+\sigma_{J\setminus S}+1}}{(\sigma_S + 1) \times (\sigma_{J\setminus S} + 1)} \\
&= \frac{|\mathcal{D}|}{\sigma_S + \sigma_{J\setminus S} + 1}.
\end{aligned}$$

From this, we may infer that $\frac{|\mathcal{D}|}{\tau} - 1 < \sigma_S + \sigma_{J\setminus S}$. Furthermore, due to the inclusion-exclusion principle, it holds that

$$\sigma_S + \sigma_{J\setminus S} - |\mathcal{D}| \leqslant \sigma_J$$

And thus, we can lower bound σ_J in terms of τ:

$$\frac{|\mathcal{D}|}{\tau} - 1 - |\mathcal{D}| \leqslant \sigma_J.$$

On the other hand, Proposition 10 gives us an upper bound on the support of any forbidden itemset. If itemset J becomes τ-forbidden in $\mathcal{D}'_k$, then it must hold that:

$$\sigma_J + 1 \leqslant |\mathcal{D}| \times (\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1).$$

We can now use both the upper and lower bound on σ_J to derive a lower bound on τ:

$$\begin{aligned}\frac{|\mathcal{D}|}{\tau} - |\mathcal{D}| - 1 &\leqslant |\mathcal{D}| \times (\frac{2}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - 1) - 1 \\ &= \frac{2|\mathcal{D}|}{\tau} - 2|\mathcal{D}|\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} - |\mathcal{D}| - 1\end{aligned}$$

or,

$$\frac{|\mathcal{D}|}{\tau} \leqslant \frac{2|\mathcal{D}|}{\tau} - 2|\mathcal{D}|\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}},$$

which in turn holds when

$$0 \leqslant \frac{|\mathcal{D}|}{\tau} - 2|\mathcal{D}|\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} = |\mathcal{D}|(\frac{1}{\tau} - 2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}}).$$

Hence, we may infer that $2\sqrt{\frac{1}{\tau^2} - \frac{1}{\tau}} \leqslant \frac{1}{\tau}$. Squaring both sides and multiplying by τ^2, we obtain that $4(1-\tau) \leqslant 1$ and thus $3/4 \leqslant \tau$. This contradicts our earlier assumption that $\tau < 3/4$. Hence, no J, S and $J \setminus S$ exist such that $\tau < \frac{|D| \times \sigma_J}{\sigma_S \times \sigma_{J \setminus S}}$ but $\frac{|\mathcal{D}|(\sigma_J+1)}{(\sigma_S+1)\times(\sigma_{J \setminus S}+1)} \leqslant \tau$ hold, for $\tau < 3/4$, as desired.

Bibliography

[1] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. DFD: Efficient functional dependency discovery. In *CIKM*, pages 949–958. ACM, 2014.

[2] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[4] Charu C Aggarwal. Outlier analysis. In *Data mining*, pages 237–263. Springer, 2015.

[5] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Record*, 22(2):207–216, 1993.

[6] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, A. Inkeri Verkamo, et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328, 1996.

[7] Abdallah Arioua and Angela Bonifati. User-guided repairing of inconsistent knowledge bases. In *EDBT*, 2018.

[8] Patricia C. Arocena, Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, and Donatello Santoro. Messing up with BART: Error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2):36–47, 2015.

[9] Ahmad Assadi, Tova Milo, and Slava Novgorodov. Cleaning data with constraints and experts. In *WebDB*, pages 1:1–1:6. ACM, 2018.

[10] Jaume Baixeries, Mehdi Kaytoue, and Amedeo Napoli. Characterizing functional dependencies in formal concept analysis with pattern structures. *AMAI*, 72(1-2):129–149, 2014.

[11] Sridevi Baskaran, Alexander Keller, Fei Chiang, Lukasz Golab, and Jaroslaw Szlichta. Efficient discovery of ontology functional dependencies. In *CIKM*, pages 1847–1856. ACM, 2017.

[12] Ricardo Bayardo, Jr., Bart Goethals, and Mohammed J. Zaki, editors. *FIMI*, 2004.

[13] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the chase. In *PODS*, pages 37–52. ACM, 2017.

[14] Roel Bertens, Jilles Vreeken, and Arno Siebes. Efficiently discovering unexpected pattern-co-occurrences. In *SDM*, pages 126–134. SIAM, 2017.

[15] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. Discovery of genuine functional dependencies from relational data with missing values. *PVLDB*, 11(8): 880–892, 2018.

[16] George Beskales, Ihab F. Ilyas, Lukasz Golab, and Artur Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552. IEEE, 2013.

[17] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. Efficient denial constraint discovery with hydra. *PVLDB*, 11(3):311–323, 2017.

[18] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154. ACM, 2005.

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

[20] Shyam Boriah, Varun Chandola, and Vipin Kumar. Similarity measures for categorical data: A comparative evaluation. In *SDM*, pages 243–254. SIAM, 2008.

[21] Jean-Francois Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queries by means of free-sets. In *PKDD*, pages 75–85. Springer, 2000.

[22] Loreto Bravo, Wenfei Fan, Floris Geerts, and Shuai Ma. Increasing the expressivity of conditional functional dependencies without extra complexity. In *ICDE*. IEEE, 2008.

[23] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD*, pages 255–264. ACM, 1997.

[24] Peter Buneman, James Cheney, Wang-Chiew Tan, and Stijn Vansummeren. Curated databases. In *PODS*, pages 1–12. ACM, 2008.

[25] Toon Calders and Bart Goethals. Depth-first non-derivable itemset mining. In *SDM*, pages 250–261. SIAM, 2005.

[26] Anup Chalamalla, Ihab F. Ilyas, Mourad Ouzzani, and Paolo Papotti. Descriptive and prescriptive data cleaning. In *SIGMOD*, pages 445–456. ACM, 2014.

[27] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys*, 41(3), 2009.

[28] Fei Chiang. *Data Quality Through Active Constraint Discovery and Maintenance*. PhD thesis, University of Toronto (Canada), 2012.

[29] Fei Chiang and Renée J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

[30] Fei Chiang and Renée J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457. IEEE, 2011.

[31] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469. IEEE, 2013.

[32] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[33] Xu Chu, Ihab F. Ilyas, Paolo Papotti, and Yin Ye. Ruleminer: Data quality rules discovery. In *ICDE*, pages 1222–1225. IEEE, 2014.

[34] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, pages 1247–1261. ACM, 2015.

[35] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *SIGMOD*, pages 2201–2206. ACM, 2016.

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

[37] Bruno Crémilleux, Arnaud Giacometti, and Arnaud Soulet. How your supporters and opponents define your interestingness. In *ECML PKDD*, page TBD. Springer, 2018.

[38] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. Nadeef: a commodity data cleaning system. In *SIGMOD*, pages 541–552. ACM, 2013.

[39] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[40] Thierno Diallo, Nöel Novelli, and Jean-Marc Petit. Discovering (frequent) constant conditional functional dependencies. *IJDMMM*, 4(3): 205–223, 2012.

[41] Amr Ebaid, Ahmed Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. Nadeef: a generalized data cleaning system. *PVLDB*, 6(12):1218–1221, 2013.

[42] Wayne W. Eckerson. Achieving business success through a commitment to high quality data. *Technical report, Data Warehousing Institute*, 2002.

[43] Wenfei Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170. ACM, 2008.

[44] Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[45] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM TODS*, 33(2), 2008.

[46] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE TKDE*, 23(5):683–698, 2011.

[47] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. Towards certain fixes with editing rules and master data. *VLDBJ*, 21(2): 213–238, 2012.

[48] Ivan P. Fellegi and David Holt. A systematic approach to automatic edit and imputation. *Journal of the American Statistical association*, 71(353):17–35, 1976.

[49] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian Saita. *Declarative data cleaning: Language, model, and algorithms*. PhD thesis, INRIA, 2001.

[50] Bernhard Ganter and Rudolf Wille. Formal concept analysis. *Wissenschaftliche Zeitschrift-Technischen Universitat Dresden*, 45:8–13, 1996.

[51] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.

[52] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. That's all folks!: LLUNATIC goes open source. *PVLDB*, 7(13):1565–1568, 2014.

[53] Bart Goethals. Survey on frequent pattern mining. *Univ. of Helsinki*, 19:840–852, 2003.

[54] Bart Goethals, Wim Le Page, and Heikki Mannila. Mining association rules of simple conjunctive queries. In *SDM*, pages 96–107. SIAM, 2008.

[55] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.

[56] Rohit Gupta, Gang Fang, Blayne Field, Michael Steinbach, and Vipin Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *KDD*, pages 301–309. ACM, 2008.

[57] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *SIGMOD Record*, 29(2):1–12, 2000.

[58] Jian He, Enzo Veltri, Donatello Santoro, Guoliang Li, Giansalvatore Mecca, Paolo Papotti, and Nan Tang. Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples. In *SIGMOD*, pages 893–907. ACM, 2016.

[59] Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data Quality and Record Linkage Techniques*. Springer, 2007.

[60] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

[61] Ihab F. Ilyas and Xu Chu. *Trends in Cleaning Relational Data: Consistency and Deduplication*. Now Publishers Inc., 2015.

[62] InsightSquared. 7 facts about data quality [infographic], 2012. URL `http://www.insightsquared.com/2012/01/7-facts-about-data-quality-infographic/`.

[63] Matteo Interlandi and Nan Tang. Proof positive and negative in data cleaning. In *ICDE*, pages 18–29. IEEE, 2015.

[64] Ariel Jarovsky, Tova Milo, Slava Novgorodov, and Wang-Chiew Tan. Rule sharing for fraud detection via adaptation. In *ICDE*. IEEE, 2018.

[65] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230. ACM, 2015.

[66] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1): 129–149, 1995.

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

[68] Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. Metric functional dependencies. In *ICDE*, pages 1275–1278. IEEE, 2009.

[69] Sebastian Kruse and Felix Naumann. Efficient discovery of approximate dependencies. *PVLDB*, 11(7):759–772, 2018.

[70] Selasi Kwashie, Jixue Liu, Jiuyong Li, and Feiyue Ye. Conditional differential dependencies (CDDs). In *ADBIS*, pages 3–17, 2015.

[71] Jiuyong Li, Jixue Liu, Hannu Toivonen, and Jianming Yong. Effective pruning for the discovery of conditional functional dependencies. *The Computer Journal*, 56(3):378–392, 2013.

[72] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. Discover dependencies from data - A review. *IEEE TKDE*, 24(2):251–264, 2012.

[73] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Functional and approximate dependency mining: database and fca points of view. *JETAI*, 14(2-3):93–114, 2002.

[74] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM TODS*, 4(4):455–469, 1979.

[75] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. Discovering reliable approximate functional dependencies. In *KDD*, pages 355–363. ACM, 2017.

[76] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *DMKD*, 1(3):241–258, 1997.

[77] Markos Markou and Sameer Singh. Novelty detection: a review. *Signal processing*, 83(12):2481–2497, 2003.

[78] Mirjana Mazuran, Elisa Quintarelli, Letizia Tanca, and Stefania Ugolini. Semi-automatic support for evolving functional dependencies. In *EDBT*, pages 293–304, 2016.

[79] Raoul Medina and Lhouari Nourine. A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In *ICFCA*, pages 98–113. Springer, 2009.

[80] Raoul Medina and Lhouari Nourine. Conditional functional dependencies: An FCA point of view. In *ICFCA*, pages 161–176. Springer, 2010.

[81] Venkata Vamsikrishna Meduri and Paolo Papotti. Towards user-aware rule discovery. In *ISIP*, pages 3–17. Springer, 2017.

[82] Noël Novelli and Rosine Cicchetti. Fun: An efficient algorithm for mining functional and embedded dependencies. In *ICDT*, pages 189–203, 2001.

[83] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Rudik: Rule discovery in knowledge bases. *PVLDB*, 11(12), 2018.

[84] Thorsten Papenbrock and Felix Naumann. A hybrid approach to functional dependency discovery. In *SIGMOD*, pages 821–833. ACM, 2016.

[85] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB*, 8(10):1082–1093, 2015.

[86] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416, 1999.

[87] Jian Pei, Anthony KH Tung, and Jiawei Han. Fault-tolerant frequent pattern mining: Problems and challenges. *DMKD*, 1:42, 2001.

[88] Clément Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam 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.

[89] J. Ross Quinlan. Induction of Decison Trees. *Machine Learning*, 1:81–106, 1986.

[90] Joeri Rammelaere. Revisiting conditional functional dependency discovery [source code], June 2018. URL https://codeocean.com/2018/06/20/discovering-conditional-functional-dependencies.

[91] Joeri Rammelaere. Cleaning data with forbidden itemsets [source code], September 2018. URL https://codeocean.com/2018/09/13/cleaning-data-with-forbidden-itemsets.

[92] Joeri Rammelaere. Xplode: Explaining repaired data with cfds [source code], June 2018. URL https://codeocean.com/2018/06/10/xplode-colon-explaining-repaired-data-with-cfds.

[93] Joeri Rammelaere and Floris Geerts. Revisiting conditional functional dependency discovery: Splitting the "C" from the "FD". In *ECML PKDD*, page TBD. Springer, 2018.

[94] Joeri Rammelaere and Floris Geerts. Explaining repaired data with CFDs. *PVLDB*, 11(11):1387–1399, 2018.

[95] Joeri Rammelaere, Floris Geerts, and Bart Goethals. Cleaning data with forbidden itemsets. In *ICDE*, pages 897–908. IEEE, 2017.

[96] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10 (11):1190–1201, 2017.

[97] Harvard Business Review. Data scientist: The sexiest job of the 21st century, 2012. URL https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century.

[98] El Kindi Rezig, Mourad Ouzzani, Walid Aref, Ahmed Elmagarmid, and Ahmed Mahmood. Pattern-driven data cleaning. *arXiv preprint arXiv:1712.09437*, 2017.

[99] Steven Ruggles and Matthew Sobek et. al. Integrated public use microdata series: Version 2.0, 1997. URL http://www.ipums.umn.edu/.

[100] Shaoxu Song and Lei Chen. Differential dependencies: Reasoning and discovery. *ACM TODS*, 36(3):16:1–16:41, 2011.

[101] Laszlo Szathmary, Petko Valtchev, Amedeo Napoli, and Robert Godin. Efficient vertical mining of frequent closures and generators. In *Advances in Intelligent Data Analysis VIII*, pages 393–404. Springer, 2009.

[102] Saravanan Thirumuruganathan, Laure Berti-Équille, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, and Nan Tang. UGuide: User-guided discovery of FD-detectable errors. In *SIGMOD*, pages 1385–1397. ACM, 2017.

[103] Maksims Volkovs, Fei Chiang, Jaroslaw Szlichta, and Renée J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255. IEEE, 2014.

[104] Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468. ACM, 2014.

[105] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[106] Yihan Wang, Shaoxu Song, Lei Chen, Jeffrey Yu, and Hong Cheng. Discovering conditional matching rules. *ACM TKDD*, 11(4):1–38, 2017.

[107] Geoffrey I. Webb. Efficient search for association rules. In *KDD*, pages 99–107. ACM, 2000.

[108] Geoffrey I. Webb and Jilles Vreeken. Efficient discovery of the most interesting associations. *ACM TKDD*, 8(3):1–31, 2014.

[109] Jef Wijsen, Raymond T. Ng, and Toon Calders. Discovering roll-up dependencies. In *KDD*, pages 213–222. ACM, 1999.

[110] Mohamed Yakout, Ahmed Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.

[111] Mohamed Yakout, Laure Berti-Équille, and Ahmed Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, pages 553–564. ACM, 2013.

[112] Hong Yao, Howard J. Hamilton, and Cory J. Butz. FD_Mine: discovering functional dependencies in a database using equivalences. In *ICDM*, pages 729–732. IEEE, 2002.

[113] I-Cheng Yeh and Che hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2, Part 1):2473 – 2480, 2009.

[114] Mohammed J. Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM*, pages 457–473. SIAM, 2002.

[115] Mohammed J. Zaki and Wagner Meira Jr. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[116] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286. ACM, 1997.