Revisiting Conditional Functional Dependency Discovery: Splitting the "C" from the "FD".

Joeri Rammelaere and Floris Geerts

University of Antwerp, Belgium
{joeri.rammelaere,floris.geerts}@uantwerp.be

Abstract. Many techniques for cleaning dirty data are based on enforcing some set of integrity constraints. Conditional functional dependencies (CFDs) are a combination of traditional Functional dependencies (FDs) and association rules, and are widely used as a constraint formalism for data cleaning. However, the discovery of such CFDs has received limited attention. In this paper, we regard CFDs as an extension of association rules, and present three general methodologies for (approximate) CFD discovery, each using a different way of combining pattern mining for discovering the conditions (the "C" in CFD) with FD discovery. We discuss how existing algorithms fit into these three methodologies, and introduce new techniques to improve the discovery process. We show that the right choice of methodology improves performance over the traditional CFD discovery method CTane.

1 Introduction

Many organizations are faced with problems arising from poor data quality, such as inaccurate or inconsistent values. In order to clean such dirty data, many techniques make use of logical rules called integrity constraints, such that values are dirty if and only if they violate a rule. These constraints are typically supplied by human experts, or discovered from the data by algorithms. Dedicated repair algorithms then modify the data such that all constraints are satisfied. In this paper we focus on the *automatic discovery of constraints*.

Among the variety of proposed constraints, Conditional functional dependencies (CFDs) have been used extensively for data cleaning. Such CFDs are a generalization of traditional functional dependencies (FDs) and association rules (ARs). CFDs are more flexible than FDs, since they can capture dependencies that hold only on a subset of the data, and more expressive and succinct than ARs, since a CFD can also identify associations that hold on the attribute level.

To discover CFDs for data cleaning, when typically only dirty data is available, it is necessary to discover *approximate* CFDs. That is, to discover CFDs that allow a certain amount of violations, in line with discovering confident association rules. To discover approximate CFDs, two algorithms have been proposed, based on the concept of *equivalence partitions*: CTane [1] and an unnamed method which we dub FindCFD [2]. These algorithms combine existing techniques for discovering FDs and ARs. While research on discovering traditional FDs has resurged in recent years, especially in the database community,

the discovery of approximate CFDs has received less attention.

In this paper, we recast CFDs as an extension of association rules, and discuss CFD discovery from a more general perspective. We 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 [1], 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 [2]. 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.

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*. We also 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. Both CTane and FindCFD were originally presented using a breadth-first strategy, because of pruning.

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.

2 Related Work

Conditional functional dependencies (CFDs) are widely used in the context of constraint-based data quality (see [3,4] for recent surveys). CFDs were introduced in [5] as an extension of Functional dependencies (FDs), and three discovery algorithms have been proposed since: CTane and FastCFD [1][2], and FindCFD [2]. Other work considers constant CFDs only [6]. Each of these discovery methods is rooted in FD discovery. Our three general approaches to CFD discovery can incorporate any FD discovery method making use of equivalence parti-

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

[2] FastCFD does not support the discovery of *approximate* CFDs

tions, e.g., Tane [7], FUN [8], FD_Mine [9], and DFD [10]. Such methods support the discovery of *approximate* dependencies, and are well suited for integration with pattern mining. An overview and experimental evaluation of functional dependency discovery is presented in [11], where it is shown that Tane is the most performant algorithm on a considerable range of data sizes. CFD discovery can also be viewed as the discovery of special conjunctive queries [12], but at the cost of a more time-consuming discovery process.

Although interesting measures for FDs based on statistical tests have recently been proposed [13], we consider approximate CFDs defined in terms of support and confidence as these are most widely used in the data quality context.

Association rules (ARs) were first introduced in [14] for supermarket basket analysis. Discovery of ARs is based on mining frequent patterns, which 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 [15]. 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, as shown in the following sections. For an overview of itemset and association rule mining, we refer to [16]. We view CFDs as a kind of ARs. An in-depth discussion relating FDs, CFDs, and ARs can be found in [17].

3 Preliminaries

We consider a relation schema R defined over a set $\mathcal{A}$ of attributes, where each attribute $\mathsf{A} \in \mathcal{A}$ has a finite domain $\mathsf{dom}(\mathsf{A})$. For an instance D of R, and tuple $t \in D$, we denote the projection of t onto a set of attributes X by $t[\mathsf{X}]$. Each tuple $t \in D$ is assumed to have a unique identifier tid, e.g., a natural number.

A *conditional functional dependency* (CFD) [5] φ over 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); and (iii) t_p is a *pattern tuple* with attributes in X and A, where for each B in $\mathsf{X} \cup \{\mathsf{A}\}$, $t_p[\mathsf{B}]$ is either a constant 'b' in $\mathsf{dom}(\mathsf{B})$, or an unnamed variable '_'. A CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ in which $t_p[\mathsf{A}] =$ '_' is called *variable*, otherwise it is *constant*. For constant CFDs, $t_p[\mathsf{X}]$ consists of constants only. Such a constant CFD is equivalent to a traditional association rule, and an FD is a CFD with t_p consisting solely of variables '_'.

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

We present CFD discovery algorithms in this paper using concepts from itemset mining. We consider *itemsets* as sets of attribute-value pairs of the form

(A, v), with $\mathsf{A} \in \mathcal{A}$, and v a value in $\mathsf{dom}(\mathsf{A})$ or '_'. An instance D thus corresponds to a *transaction* database, with each tuple corresponding to a transaction of length $|\mathcal{A}|$. An item (A, v) with $v \in \mathsf{dom}(\mathsf{A})$ is *supported* in a tuple t if $t[\mathsf{A}] = v$. Items $(\mathsf{A}, \_)$ are supported by every transaction. A tuple supports an *itemset* I in D if it supports all items $i \in I$. The *cover* of an itemset I in D, denoted by $\mathsf{cov}(I, D)$ and also called I's *tidlist*, is the set of tids of tuples in D that support I. The *support* of I in D, denoted by $\mathsf{supp}(I, D)$, is equal to the number of tids in I's cover in D.

We can now write a CFD $\varphi = (\mathsf{X} \to \mathsf{A}, t_p)$ compactly as an *association rule* $I \to j$, between an itemset I and a single item j, where $I = \bigcup_{\mathsf{B} \in \mathsf{X}} \{(\mathsf{B}, t_p[\mathsf{B}])\}$ and $j = (\mathsf{A}, t_p[\mathsf{A}])$. In line with the notion of approximate FDs [7], we define the *confidence* of a CFD $\varphi = I \to j$ as $\mathsf{conf}(\varphi, D) = 1 - \frac{|D'|}{\mathsf{supp}(I,D)}$, where $D' \subset D$ is a minimal subset such that $D \setminus D' \models \varphi$. For a constant CFD, $|D'| = |\mathsf{VIO}(\varphi, D)|$, and hence $\mathsf{conf}(\varphi, D) = (\mathsf{supp}(I, D) - |\mathsf{VIO}(\varphi, D)|)/\mathsf{supp}(I, D) = \mathsf{supp}(I \cup \{j\}, D)/\mathsf{supp}(I, D)$ reduces to the standard confidence of an association rule. For variable CFDs, $|D'|$ is the minimum number of tuples that need to be altered or removed for φ to be satisfied. For example, 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. A CFD φ is called *exact* if $\mathsf{conf}(\varphi, D) = 1$, and *approximate* otherwise.

Finally, we consider CFD discovery algorithms based on the concept of *equivalence partitions*, as used in the Tane algorithm [7]. 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 $s \in D$ [3]. 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, $\Pi((\mathsf{A}, \_)) = \{\mathsf{cov}((\mathsf{A}, v)) \mid v \in \mathsf{dom}(\mathsf{A})\}$, 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 iff $|\Pi(I)| = |\Pi(I \cup \{j\})|$.

Problem Statement. *Given an instance D of a schema R, support threshold δ, and confidence threshold ε, the* approximate CFD discovery problem *is to find all CFDs φ over R with* $\mathsf{supp}(\varphi, D) \geq \delta$ *and* $\mathsf{conf}(\varphi, D) \geq 1 - \varepsilon$.

Example 1. We use the "play tennis" dataset from [18], shown in Table 1. One of the approximate CFDs φ on this dataset is $\big\{(\mathsf{Windy}, \text{false}), (\mathsf{Outlook}, \_)\big\} \to (\mathsf{Play}, \_)$. Let $I = \big\{(\mathsf{Windy}, \text{false}), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\big\}$ and $j = (\mathsf{Play}, \_)$. The relevant equivalence partitions are $\Pi(I \setminus \{j\}) = \big\{\{1, 8, 9\}, \{3, 13\}, \{4, 5, 10\}\big\}$ and $\Pi(I) = \big\{\{1, 9\}, \{8\}, \{3, 13\}, \{4, 5, 10\}\big\}$. The sizes of the equivalence partitions

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

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 $t[\mathsf{Windy}] = \text{false}$, are shaded grey in Table 1, with different shades 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 $\Pi(I \setminus \{j\}) = \Pi(I)$, and hence its confidence is $1 - (|D'|/||\Pi(I)||) = 1 - (1/8) = 0.875$. Finally, $\mathsf{VIO}(\varphi, D) = \{t_1, t_8, t_9\}$. $\diamondsuit$

Table 1. Running example based on the play tennis dataset [18]

tid	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	dont
2	sunny	hot	high	true	dont
3	overcast	hot	high	false	play
4	rain	mild	high	false	play
5	rain	cool	normal	false	play
6	rain	cool	normal	true	dont
7	overcast	cool	normal	true	play
8	sunny	mild	high	false	dont
9	sunny	cool	normal	false	play
10	rain	mild	normal	false	play
11	sunny	mild	normal	true	play
12	overcast	mild	high	true	play
13	overcast	hot	normal	false	play
14	rain	mild	high	true	dont

4 Three approaches for CFD Discovery

We present three general approaches for the discovery of approximate CFDs with high supports. These approaches differ in the way that the (itemset) search lattice is explored. First, we generalize the *integrated* approach [1], in which the 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, we describe our algorithms *independent* from the search strategy used. To achieve uniform pruning across all approaches and search strategies, we present pruning strategies based on a generalization of free itemsets [19] and a lookup table.

4.1 Integrated CFD discovery

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

Algorithm 1 Integrated CFD discovery algorithm

```
1: procedure MINE-INTEGRATED(D, δ, ε)
2:     L ← {(A, v) | A ∈ A, v ∈ dom(A) ∪ {_}, supp((A, v), D) ≥ δ}
3:     Compute Π({i}, D) for all i ∈ L
4:     Initialize fringe with L 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(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
```

gorithms based on this methodology traverse the entire search lattice for CFDs, consisting of both constant *and* variable patterns. The first level $\mathcal{L}$ of this lattice is initialized on line 2. For each singleton item, its equivalence partition is computed from the data; only sufficiently frequent constant items are retained.

The lattice is subsequently traversed, typically in either a breadth-first or depth-first manner [4]. 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 4), 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 [7], and is based on the *error* of an equivalence class. More precisely, for all $\mathsf{eq} \in \Pi(I \setminus \{j\})$, let $\Pi(I)^{\mathsf{eq}}$ denote those $\mathsf{eq}' \in \Pi(I)$ with $\mathsf{eq}' \subset \mathsf{eq}$. In other words, $\Pi(I)^{\mathsf{eq}}$ contains all equivalence classes over I that match the same (constant) pattern as eq on the attributes $I \setminus \{j\}$. We define

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

Generalizing the argument given in [7] for variable patterns to arbitrary (constant and variable) patterns, the confidence can then be computed as:

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

Example 2. We consider the CFD $\big\{(\mathsf{Windy}, \mathrm{false}), (\mathsf{Outlook}, \_)\big\} \to (\mathsf{Play}, \_)$ from our running example, and let $I = \big\{(\mathsf{Windy}, \mathrm{false}), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\big\}$ and $j = (\mathsf{Play}, \_)$. We compute the error for each of the 3 equivalence classes in $\Pi(I \setminus \{j\}) = \big\{\{1, 8, 9\}, \{3, 13\}, \{4, 5, 10\}\big\}$. For $\mathsf{eq} = \{3, 13\}$ and $\mathsf{eq} = \{4, 5, 10\}$, we have $|\Pi(I)^{\mathsf{eq}}| = 1$, since the tuples within these equivalence classes have the same values for attribute Play. Hence, there is only one $\mathsf{eq}' \in \Pi(I)^{\mathsf{eq}}$, and $||\Pi(I)^{\mathsf{eq}}|| =$

[4] The CTane algorithm as presented in [1] employs a breadth-first traversal.

$\max_{\mathsf{eq}' \in \Pi(I)^{\mathsf{eq}}} |\mathsf{eq}'|$, leading to an error of 0. This leaves us with $\mathsf{eq} = \{1, 8, 9\}$, for which $\Pi(I)^{\mathsf{eq}} = \big\{\{1, 9\}, \{8\}\big\}$. Indeed, this is the equivalence class containing the violations of the CFD. We compute the error as $\mathsf{error} = ||\big\{\{1, 9\}, \{8\}\big\}|| - \max(|\{1, 9\}|, |\{8\}|) = 1$, resulting in a confidence of $1 - (\mathsf{error}/||\Pi(I)||) = 1 - (1/8) = 0.875$, as mentioned in Ex. 1. ♢

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 (see details in the appendix).

4.2 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 Alg. 2. The search lattice $\mathcal{L}$ is initialized (line 2) using only items with constant values. We therefore only require the cover of each item in $\mathcal{L}$ (the equivalence partition of a constant item corresponds to its cover). The lattice is traversed using an arbitrary search strategy and generated itemsets are inserted into the fringe.

When visiting itemset I in this approach, we initialize a separate FD searching algorithm (line 8). The item lattice for this FD search ($\mathcal{L}^{\mathrm{FD}}$) now consists only of those items in D with a variable pattern ('_'), and whose attribute is not already present in $\mathsf{attrs}(I)$, the set of attributes in the items in I. In other words, we wish to extend the constant pattern I with variable patterns to obtain CFDs. During the traversal of $\mathcal{L}^{\mathrm{FD}}$ the equivalence partition of each item is computed on D^I, the dataset D projected on I, i.e., using only those tuples with a tid in $\mathsf{cov}(I, D)$. The algorithm Find-FDs is then invoked (line 10), which can be any FD-discovery algorithm using equivalence partitions, to discover all FDs with confidence $\geq 1 - \varepsilon$ on D^I. The resulting FDs are augmented with the pattern I, and added to the set Σ of CFDs (line 11). Note that Find-FDs is oblivious to the support threshold δ, since an FD is supported by all tuples in D^I, and $|D^I| \geq \delta$ is already ensured by enforcing the support threshold on I. Pseudocode of Find-FDs is available in the appendix.

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

Algorithm 2 Itemset-First CFD discovery algorithm

```
1: procedure MINE-ITEMSET-FIRST(D, δ, ε)
2:     L ← {(A, v) | A ∈ A, v ∈ dom(A), supp((A, v), D) ≥ δ}
3:     Compute cov({i}, D) for all i ∈ L
4:     Initialize fringe with L depending on search strategy
5:     Σ ← ∅
6:     while fringe not empty do
7:         I ← POP(fringe)
8:         L^FD ← {(A, _) | A ∈ A \ attrs(I)}
9:         Compute Π({k}, D^I) for all k ∈ L^FD
10:        Σ^FD ← FIND-FDS(L^FD, D^I, I, ε)
11:        Σ ← Σ ∪ {I ∪ J → j | J → j ∈ Σ^FD}
12:        insert children of I into fringe if their support ≥ δ
13:    return Σ
```

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 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 $\mathsf{cov}(I, D)$ and $\mathsf{cov}(J, D)$, for the itemsets J in the fringe with $|J| = |I|$, and such that J and I differ in only one item.

4.3 FD-First discovery

The third and final approach to CFD discovery, MINE-FD-FIRST, is shown in pseudocode in Alg. 3. This approach is a generalization of the FindCFD algorithm [2], which starts with FD discovery. The search lattice $\mathcal{L}$ is thus initialized (line 2) using only variable items, i.e., one item $(\mathsf{A}, \_)$ for each attribute $\mathsf{A} \in \mathcal{A}$. As before, equivalence partitions are computed, after which a fringe is created and a breadth 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, the lattice $\mathcal{L}^{\text{Pat}}$ of constant items is explored. This lattice is initialized on line 12.

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 $|\mathsf{cov}(J, \Pi(I))| \ll |\mathsf{cov}(J, D)|$, efficiency is increased.

Algorithm 3 FD-First CFD discovery algorithm

```
1: procedure MINE-FD-FIRST(D, δ, ε)
2:     L ← {(A, _) | A ∈ A}
3:     Compute Π({i}, D) for all i ∈ L
4:     Initialize fringe with L 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(I \ {j} → j, D) ≥ 1 − ε then
10:                Σ ← Σ ∪ {I \ {j} → j}
11:            if conf(I \ {j} → j, D) < 1 then
12:                L^Pat ← {(A, v) | A ∈ attrs(I), v ∈ dom(A)}
13:                Compute cov({i}, Π(I)) for all i ∈ L^Pat
14:                Σ ← Σ ∪ MINE-PATTERNS(L^Pat, I \ {j} → j, Π(I), δ, ε)
15:        insert children of I into fringe
16:    return Σ
```

Example 4. Consider the FD $\{(\mathsf{Windy}, \_), (\mathsf{Outlook}, \_)\} \to (\mathsf{Play}, \_)$ corresponding to the itemset $I = \{(\mathsf{Windy}, \_), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\}$, with equivalence class $\Pi(I) = \{\{1, 9\}, \{2\}, \{3, 13\}, \{4, 5, 10\}, \{6, 14\}, \{7, 12\}, \{8\}, \{11\}\}$. The constant pattern $(\mathsf{Windy}, \mathrm{false})$ can now be represented by its pidlist. That is, $\mathsf{cov}((\mathsf{Windy}, \mathrm{false}), \Pi(I)) = \{1, 3, 4, 7\}$. Since $\mathsf{supp}((\mathsf{Windy}, \mathrm{false}), D) = 8$, 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 $\mathsf{supp}(M, \Pi(I)) \geq \delta$ are considered as possible patterns for a CFD. Whenever an itemset M is processed in MINE-PATTERNS, we validate the CFD $(I \setminus \{j\}) \oplus M \to j$, where $\oplus$ *replaces* those variable items in $(I \setminus \{j\})$ which have a constant counterpart in M, i.e., $(I \setminus \{j\}) \oplus M = M \cup \{(\mathsf{A}, \_) \in I \setminus \{j\} \mid \mathsf{A} \notin \mathsf{attrs}(M)\}$. If the CFD is sufficiently confident, it is added to the result.

Pseudocode for algorithm MINE-PATTERNS is available in the appendix. 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 [2] takes a similar approach, but, to our knowledge, does not perform an exhaustive search through the pattern lattice, i.e., the power set of $\mathcal{L}^{\text{Pat}}$. 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 [20].

4.4 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 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 5.

4.5 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 1 (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 $(\mathsf{A}, v) \in J$, 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 5. In our running example, the itemsets $\{(\mathsf{Windy}, \text{false}), (\mathsf{Outlook}, \_)\}$ and $\{(\mathsf{Windy}, \_), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\}$, among others, are preceding sets of the itemset $\{(\mathsf{Windy}, \text{false}), (\mathsf{Outlook}, \_), (\mathsf{Play}, \_)\}$. $\diamondsuit$

Definition 2 (CFD Redundancy). *Consider a database instance D and a CFD $\varphi : I \to j$ with $\mathsf{conf}(\varphi, D) \geq 1 - \varepsilon$. 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 6. In our example, the CFD $(\mathsf{Temperature}, \text{Cool}) \to (\mathsf{Humidity}, \text{Normal})$ holds exactly. This implies the redundancy of, for example, the CFDs

$$\{(\mathsf{Temperature}, \text{Cool}), (\mathsf{Humidity}, \text{Normal}), (\mathsf{Windy}, \_)\} \to (\mathsf{Play}, \_)$$
$$\{(\mathsf{Temperature}, \text{Cool}), (\mathsf{Windy}, \_)\} \to (\mathsf{Humidity}, \_). \quad \diamondsuit$$

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 $\mathcal{C}^+(X) = \mathcal{C}^+(X) \cap \mathcal{C}^+(I)$ 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 [5], 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 $\mathcal{C}^+(I \cup \{j\}) = \mathcal{C}^+(I \cup \{j\}) \cap M$.

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

Table 2. 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
Mushroom	8124	119	15
Nursery	12960	32	9

Our second pruning technique generalizes the concept of free itemsets [19] (also called generators [21]). 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 3 (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 $\varphi : I \rightarrow j$ 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 [22].

5 Experiments

We experimentally validate the proposed techniques on real-life datasets from the UCI repository (http://archive.ics.uci.edu/ml/), described in Table 2. 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++`, the source code and used datasets are available for research purposes [6]. The program was tested on an Intel Xeon Processor (3.8GHZ) with 32GB of memory running Ubuntu. Our algorithms run entirely in main memory.

In Sec. 4, 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:

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 – out

[6] https://bit.ly/2yFNksO

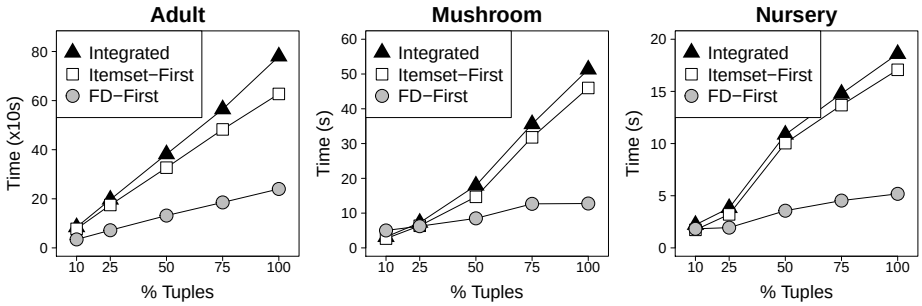

Fig. 1. Scalability of three CFD discovery algorithms in number of tuples.

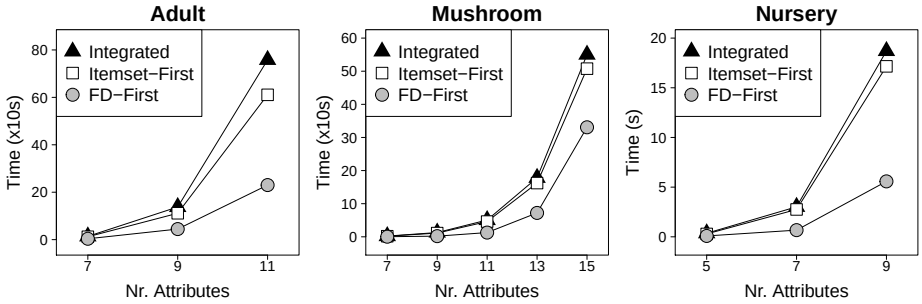

Fig. 2. Scalability of three CFD discovery algorithms in number of attributes.

of a total of 18 different combinations. The runtime results of all instantiations are available in the appendix.

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. The confidence threshold was found to have a negligible influence on runtime, and hence all experiments are run with $\varepsilon = 0$. Runtime plots in function of confidence can be found in the appendix. We emphasize that all methods return the exact same result in every experiment.

5.1 Number of Tuples

We first investigate the scalability of each approach in terms of the number of tuples. For this experiment, we consider only the first $X\%$ tuples of each dataset, with X ranging from 10% to 100%. The minimum support threshold was fixed at 10% of the number of tuples considered, and the maximum antecedent size was fixed at 6. The obtained runtimes are displayed in Fig. 1. We see that the FD-First approach scales better than the other approaches, and is faster overall.

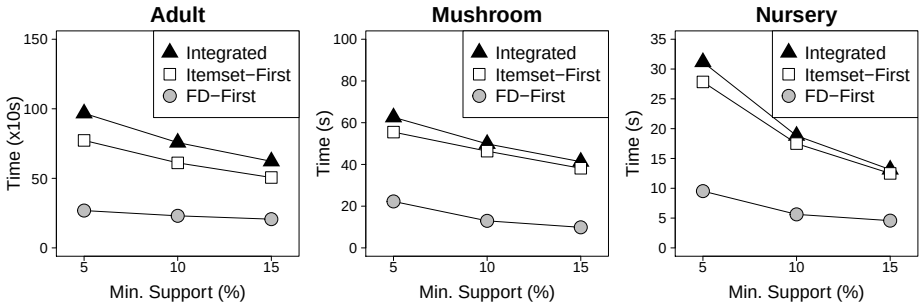

Fig. 3. Scalability of three CFD discovery algorithms in minimum support threshold.

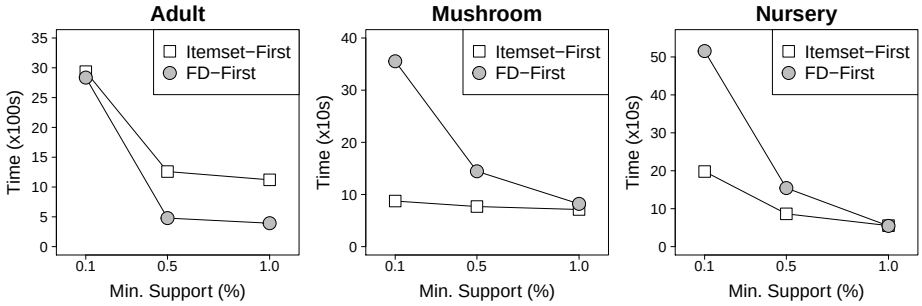

Fig. 4. Scalability of Itemset-First and FD-First discovery algorithms for very low minimum support thresholds.

5.2 Number of Attributes

Similar to the previous experiment, we now investigate the performance of the three algorithms in terms of the number of attributes, by considering only the first X attributes. In Fig. 2, 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. While each of the algorithms shows an exponential rise in runtime as the number of attributes increases, the FD-First method clearly outperforms the other approaches. The Integrated method is the slowest overall, and suffers most of all from the increasing number of attributes.

5.3 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 Fig. 3, for minimum support thresholds of 5%, 10%, and 15% of the total number of tuples. Overall, the support threshold has less impact than the number of 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 somewhat similar increase.

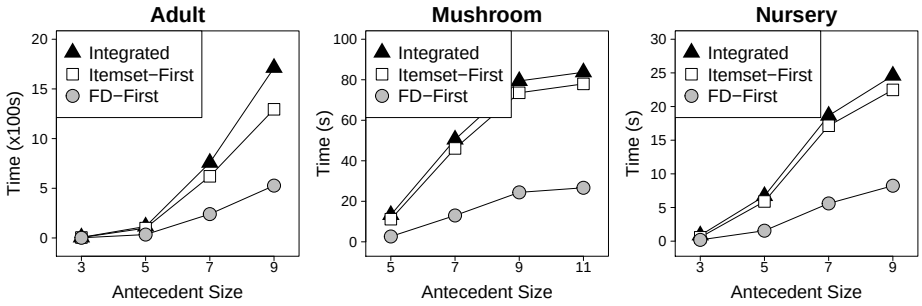

Fig. 5. Scalability of three CFD discovery algorithms in maximal size of antecedent.

However, the situation changes when considering very low support thresholds. In Fig. 4, 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 support 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.

5.4 Maximal Antecedent Size

We conclude the experimental section by investigating the impact of the maximal antecedent size threshold on the runtime of the algorithms. The results are shown in Fig. 5. The minimum support threshold was again fixed at 10%. We see an exponential increase in runtime, similar to that observed when the number of attributes was increased. The FD-First approach again performs best on every dataset, and shows the lowest increase in runtime as antecedent size increases.

6 Conclusion

We have presented the discovery of Conditional functional dependencies (CFDs) as a form of association rule mining, and classified the possible approaches into three categories, based on how these approaches combine pattern mining and functional dependency discovery. Two of these approaches have not been considered before. Moreover, we discuss how 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 outperform the existing CTane algorithm, and identify situations in which either of these methods achieve the best performance. 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. As future work, we plan to investigate parallelized or distributed discovery and develop incremental discovery methods to accommodate for dynamic, changing data.

References

1. W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *TKDE*, vol. 23, no. 5, pp. 683–698, 2011.
2. F. Chiang and R. J. Miller, "Discovering data quality rules," *PVLDB*, vol. 1, no. 1, pp. 1166–1177, 2008.
3. W. Fan and F. Geerts, *Foundations of Data Quality Management*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
4. I. F. Ilyas and X. Chu, "Trends in cleaning relational data: Consistency and deduplication," *Foundations and Trends in Databases*, vol. 5, no. 4, pp. 281–393, 2015.
5. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *TODS*, vol. 33, no. 2, 2008.
6. T. Diallo, N. Novelli, and J.-M. Petit, "Discovering (frequent) constant conditional functional dependencies," *IJDMMM*, vol. 4, no. 3, pp. 205–223, 2012.
7. Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: an efficient algorithm for discovering functional and approximate dependencies," *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.
8. N. Novelli and R. Cicchetti, "Fun: An efficient algorithm for mining functional and embedded dependencies," in *ICDT*. Springer, 2001, pp. 189–203.
9. H. Yao, H. J. Hamilton, and C. J. Butz, "Fd_mine: discovering functional dependencies in a database using equivalences," in *ICDM*. IEEE, 2002, pp. 729–732.
10. Z. Abedjan, P. Schulze, and F. Naumann, "Dfd: Efficient functional dependency discovery," in *CIKM*. ACM, 2014, pp. 949–958.
11. T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann, "Functional dependency discovery: An experimental evaluation of seven algorithms," *PVLDB*, vol. 8, no. 10, pp. 1082–1093, 2015.
12. B. Goethals, W. L. Page, and H. Mannila, "Mining association rules of simple conjunctive queries," in *SDM*. SIAM, 2008, pp. 96–107.
13. P. Mandros, M. Boley, and J. Vreeken, "Discovering reliable approximate functional dependencies," in *KDD*. ACM, 2017, pp. 355–363.
14. R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Acm sigmod record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.
15. M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li *et al.*, "New algorithms for fast discovery of association rules." *KDD*, pp. 283–286, 1997.
16. M. J. Zaki and W. Meira Jr, *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.
17. R. Medina and L. Nourine, "A unified hierarchy for functional dependencies, conditional functional dependencies and association rules," in *ICFCA*. Springer, 2009, pp. 98–113.
18. J. R. Quinlan, "Induction of Decison Trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.
19. J.-F. Boulicaut, A. Bykowski, and C. Rigotti, "Approximation of frequency queries by means of free-sets," in *PKDD*. Springer, 2000, pp. 75–85.
20. F. Chiang, "Data quality through active constraint discovery and maintenance," Ph.D. dissertation, University of Toronto (Canada), 2012.
21. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *ICDT*, 1999, pp. 398–416.
22. L. Szathmary, P. Valtchev, A. Napoli, and R. Godin, "Efficient vertical mining of frequent closures and generators," in *Advances in Intelligent Data Analysis VIII*. Springer, 2009, pp. 393–404.

Appendix A: Additional Experiments

A.1 Influence of Confidence Threshold

The runtime results in function of confidence are shown in Fig. 6. 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.

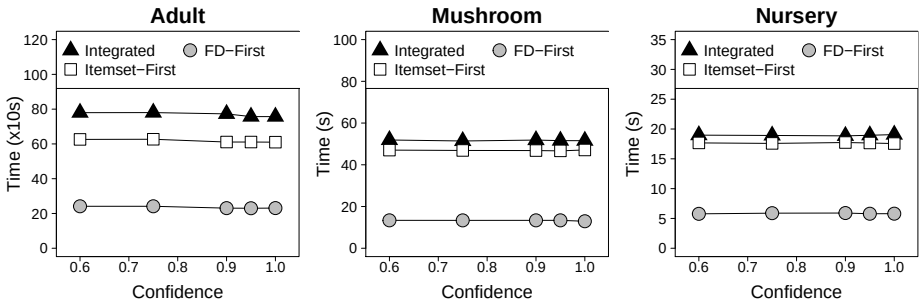

Fig. 6. Influence of confidence threshold on runtime of three CFD discovery algorithms.

A.2 Number of CFDs

As additional information, we show the number of CFDs found for various support and confidence thresholds in Table 3. 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 on the Mushroom dataset. Moreover, while the number is manageable for high confidence thresholds, CFD discovery also suffers from pattern explosion on low confidence thresholds. As noted in the experimental section, all algorithms returned the same CFDs.

A.3 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 Fig. 7. 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

Table 3. 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
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

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.

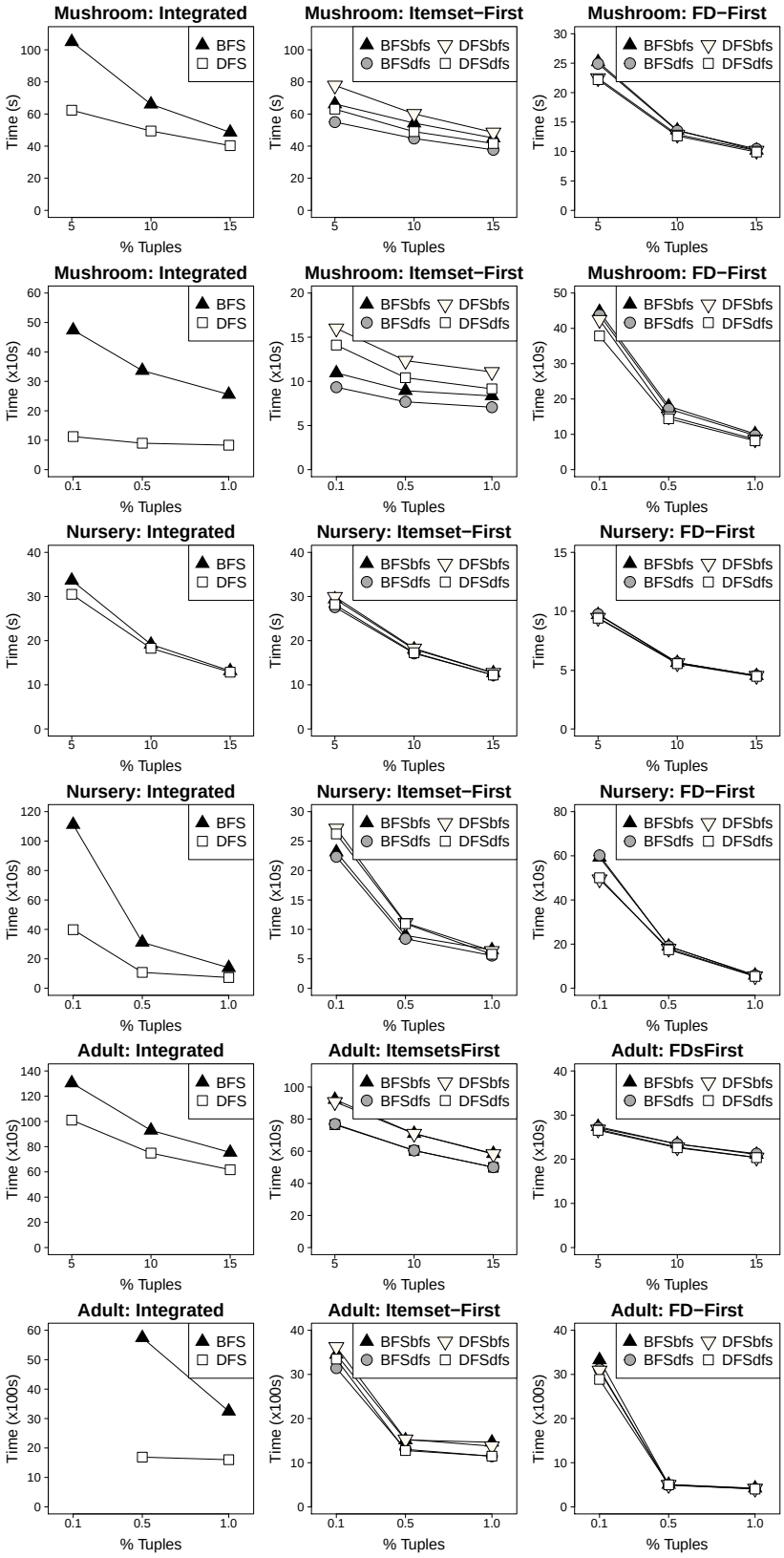

Fig. 7. Comparison of different search strategies for each of the three methodologies.

Appendix B: Pseudocodes of Algorithms

B.1 Intersection

The algorithm for computing the intersection of two equivalence partitions, $\Pi(I)$ and $\Pi(J)$, as presented in [7], is shown using our terminology in Alg. 4. 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.

Algorithm 4 Intersection algorithm for equivalence partitions

```
1: procedure INTERSECTION(D, 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)
```

B.2 Find-FDs

Pseudocode for the FIND-FDS algorithm, which is used as the second step in the Itemset-First CFD discovery method, is shown in Alg. 5. The algorithm takes as input the first level in the lattice, the projected database D^P for a constant pattern P, and the confidence threshold ε. The lattice contains all variable items, except those in $\mathsf{attrs}(P)$, with their equivalence partitions computed on 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 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.

Algorithm 5 FD-discovery subroutine for Itemset-First algorithm

procedure FIND-FDS($\mathcal{L}, D^P, P, \varepsilon$)
  Initialize *fringe* with $\mathcal{L}$ depending on search strategy
  $\Sigma \leftarrow \emptyset$
  while *fringe* **not** empty **do**
    $I \leftarrow$ POP(*fringe*)
    for all $j \in I$ **do**
      if $\mathsf{conf}(I \setminus \{j\} \rightarrow j, D^P) \geq 1 - \varepsilon$ **then**
        $\Sigma \leftarrow \Sigma \cup (P \cup (\{I \setminus \{j\})) \rightarrow j\}$
    insert children of I into *fringe*
  return Σ

B.3 Mine-Patterns

Pseudocode for the MINE-PATTERNS algorithm, which is used as the second step in the FD-First CFD discovery method, is shown in Alg. 6. The algorithm takes as input the first level of the lattice, an FD $I \setminus \{j\} \rightarrow j$ 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. Here, the lattice contains only constant items with attributes in $\mathsf{attrs}(I \setminus \{j\})$, and their corresponding *pidlists*, i.e., their covers 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 $(I \setminus \{j\}) \oplus M \rightarrow j$ 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 6 Pattern mining subroutine for FD-First algorithm

procedure MINE-PATTERNS($\mathcal{L}, I \setminus \{j\} \rightarrow j, \Pi(I), \delta, \varepsilon$)
  Initialize *fringe* with $\mathcal{L}$ depending on search strategy
  $\Sigma \leftarrow \emptyset$
  while *fringe* **not** empty **do**
    $M \leftarrow$ POP(*fringe*)
    $cfd \leftarrow (I \setminus \{j\}) \oplus M \rightarrow j$
    if $\mathsf{conf}(cfd, \Pi(I)) \geq 1 - \varepsilon$ **then**
      $\Sigma \leftarrow \Sigma \cup cfd$
    insert children of M into *fringe* if their support $\geq \delta$
  return Σ