Discovering Conditional Functional Dependencies

Wenfei Fan , Floris Geerts , Jianzhong Li , Ming Xiong

Abstract—This paper investigates the discovery of conditional functional dependencies (CFDs). CFDs are a recent extension of functional dependencies (FDs) by supporting patterns of semantically related constants, and can be used as rules for cleaning relational data. However, finding quality CFDs is an expensive process that involves intensive manual effort. To effectively identify data cleaning rules, we develop techniques for discovering CFDs from relations. Already hard for traditional FDs, the discovery problem is more difficult for CFDs. Indeed, mining patterns in CFDs introduces new challenges. We provide three methods for CFD discovery. The first, referred to as CFDMiner, is based on techniques for mining closed itemsets, and is used to discover *constant* CFDs, namely, CFDs with constant patterns only. Constant CFDs are particularly important for object identification, which is essential to data cleaning and data integration. The other two algorithms are developed for discovering general CFDs. One algorithm, referred to as CTANE, is a levelwise algorithm that extends TANE, a well-known algorithm for mining FDs. The other, referred to as FastCFD, is based on the depth-first approach used in FastFD, a method for discovering FDs. It leverages closed-itemset mining to reduce the search space. As verified by our experimental study, CFDMiner efficiently discovers constant CFDs. For general CFDs, CTANE works well when a given relation is large, but it does not scale well with the arity of the relation. FastCFD is far more efficient than CTANE when the arity of the relation is large; better still, leveraging optimization based on closed-itemset mining, FastCFD also scales well with the size of the relation. These algorithms provide a set of cleaning-rule discovery tools for users to choose for different applications.

Index Terms—Integrity, Conditional functional dependency, Functional dependency, Free itemset, Closed itemset

1 Introduction

Conditional functional dependencies (CFDs) [1] were recently introduced for data cleaning. They extend standard functional dependencies (FDs) by enforcing patterns of semantically related constants. CFDs have been proven more effective than FDs in detecting and repairing inconsistencies (dirtiness) of data [1], [2], and are expected to be adopted by data-cleaning tools that currently employ standard FDs (*e.g.*, [3], [4], [5]; see [6], [7] for surveys on data cleaning tools).

However, for CFD-based cleaning methods to be effective in practice, it is necessary to have techniques in place that can *automatically discover* or *learn* CFDs from sample data, to be used as data cleaning rules. Indeed, it is often unrealistic to rely solely on human experts to design CFDs via an expensive and long manual process. As indicated in [8], cleaning-rule discovery is critical to commercial data quality tools.

This practical concern highlights the need for studying the *discovery problem* for CFDs: given a sample instance r of a relation schema R, it is to find a *canonical cover* of all CFDs that hold on r, *i.e.*, a set of CFDs that is logically equivalent to the set of all CFDs that hold on r. To reduce redundancy, each CFD in the canonical cover should be *minimal*, *i.e.*, *nontrivial* and *left-reduced* (see [9] for nontrivial and left-reduced FDs).

The discovery problem is, however, *highly nontrivial*. It is already hard for traditional FDs since, among other things, a canonical cover of FDs discovered from a relation r is inherently *exponential* in the *arity* of the schema of r, *i.e.*, the number of attributes in R. Since CFD discovery subsumes FD discovery, *the exponential complexity* carries over to CFD discovery. Moreover, CFD discovery requires mining of semantic patterns with constants, a challenge that was not encountered when discovering FDs, as illustrated by the example below.

Example 1: The following relational schema cust is taken from [1]. It specifies a customer in terms of the customer's phone (country code (CC), area code (AC), phone number (PN)), name (NM), and address (street (STR), city (CT), zip code (ZIP)). An instance r_0 of cust is shown in Fig. 1.

Traditional FDs that hold on r_0 include the following:

f_1: $[\mathsf{CC}, \mathsf{AC}] \rightarrow \mathsf{CT}$
f_2: $[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \rightarrow \mathsf{STR}$

Here f_1 requires that two customers with the same country- and area-codes also have the same city; similarly for f_2.

In contrast, the CFDs that hold on r_0 include not only the FDs f_1 and f_2, *but also* the following (and more):

ϕ_0: $([\mathsf{CC}, \mathsf{ZIP}] \rightarrow \mathsf{STR}, (44, \_ \parallel \_))$
ϕ_1: $([\mathsf{CC}, \mathsf{AC}] \rightarrow \mathsf{CT}, (01, 908 \parallel \text{MH}))$
ϕ_2: $([\mathsf{CC}, \mathsf{AC}] \rightarrow \mathsf{CT}, (44, 131 \parallel \text{EDI}))$
ϕ_3: $([\mathsf{CC}, \mathsf{AC}] \rightarrow \mathsf{CT}, (01, 212 \parallel \text{NYC}))$

In ϕ_0, $(44, \_ \parallel \_)$ is the pattern tuple that enforces a binding of semantically related constants for attributes (CC, ZIP, STR) in a tuple. It states that for customers in the UK, ZIP uniquely determines STR. It is an FD that only holds on the subset of tuples with the pattern "CC = 44", rather than on the entire relation r_0. CFD ϕ_1 assures that for any customer in the US (country code 01) with area code 908, the city of the customer *must* be MH, as enforced by its pattern tuple $(01, 908 \parallel \text{MH})$; similarly for ϕ_2 and ϕ_3. These cannot be expressed as FDs.

More specifically, a CFD is of the form $(X \rightarrow A,\ t_p)$, where $X \rightarrow A$ is an FD and t_p is a *pattern tuple* with attributes in X

• *W. Fan and F. Geerts are with the University of Edinburgh. Email: {wenfei,fgeerts}@inf.ed.ac.uk.*
• *J. Li is with Harbin Institute of Technology. Email: lijzh@hit.edu.cn.*
• *M. Xiong and W. Fan are with Bell Laboratories. Email: {xiong,wenfei}@research.bell-labs.com.*

	CC	AC	PN	NM	STR	CT	ZIP
t_1:	01	908	1111111	Mike	Tree Ave.	MH	07974
t_2:	01	908	1111111	Rick	Tree Ave.	MH	07974
t_3:	01	212	2222222	Joe	5th Ave	NYC	01202
t_4:	01	908	2222222	Jim	Elm Str.	MH	07974
t_5:	44	131	3333333	Ben	High St.	EDI	EH4 1DT
t_6:	44	131	4444444	Ian	High St.	EDI	EH4 1DT
t_7:	44	908	4444444	Ian	Port PI	MH	W1B 1JH
t_8:	01	131	2222222	Sean	3rd Str.	UN	01202

Fig. 1. An instance r_0 of the cust relation.

and A. The pattern tuple consists of constants and an unnamed variable '_' that matches an arbitrary value. To discover a CFD it is necessary to find not only the traditional FD $X \rightarrow A$ but also its pattern tuple t_p. With the same FD $X \rightarrow A$ there are possibly multiple CFDs defined with different pattern tuples, *e.g.,* ϕ_1–ϕ_3. Hence a canonical cover of CFDs that hold on r_0 is typically much larger than its FD counterpart. Indeed, as recently shown by [10], provided that a fixed FD $X \rightarrow A$ is already given, the problem for discovering sensible patterns associated with the FD alone is already NP-complete. □

Observe that the pattern tuple in each of ϕ_1–ϕ_3 consists of only constants in both its LHS and RHS. Such CFDs are referred to as *constant* CFDs. Constant CFDs are instance-level FDs [11] that are particularly useful in object identification, an issue essential to both data quality and data integration.

Prior work. The discovery problem has been studied for FDs for two decades [12], [13], [14], [15], [16], [17], [18], [19] for database design, data archiving, OLAP and data mining. It was first investigated in [12], which shows that the problem is inherently exponential in the arity $|R|$ of the schema R of sample data r. One of the best-known methods for FD discovery is TANE [13], a *levelwise* algorithm [20] that searches an attribute-set containment lattice and derives FDs with $k + 1$ attributes from sets of k attributes, with pruning based on FDs generated in previous levels. TANE takes linear time in the size $|r|$ of *input* sample r, and works well when the arity $|R|$ is not very large. The algorithms of [16], [17], [18] follow a similar levelwise approach. However, the levelwise algorithms may take exponential time in $|R|$ even if the output is not exponential in $|R|$. In light of this, another algorithm, referred to as FastFD [14], explores the connection between FD discovery and the problem of finding minimal covers of hypergraphs, and employs the depth-first strategy to search minimal covers. Its takes (almost) linear-time in the size of the *output*, *i.e.,* in the size of the FD cover. It scales better than TANE when the arity is large, but it is more sensitive to the size $|r|$. Indeed, it is in $O(|r|^2 \log |r|)$ time, when considering data complexity ($|R|$ is assumed constant). There has also been a bottom-up approach [15] based on techniques for learning general logical descriptions in a hypotheses space. As shown in [13], TANE outperforms the algorithm of [15].

Recently two sets of algorithms have been developed for discovering CFDs [10], [21]. For a fixed traditional FD fd, [10] showed that it is NP-complete to find useful patterns that, together with fd, make quality CFDs. They provide efficient heuristic algorithms for discovering patterns from samples *w.r.t.* a fixed FD. An algorithm for discovering CFDs, including both traditional FDs and their associated patterns, was presented in [21], which is an extension of TANE.

Constant CFD discovery is closely related to association rule mining (*e.g.,* [22]) and in particular, closed and free itemsets mining (*e.g.,* [23], [24]). With 100% confidence, an association rule $(X, t_p) \Rightarrow (A, a)$ is a constant CFD $(X \rightarrow A,\ (t_p \parallel a))$, where t_p is a constant pattern over attributes X and a is a value in the domain of attribute A. Better still, there is an intimate connection between left-reduced constant CFDs and non-redundant association rules, which can be found by computing closed itemsets and free itemsets.

The potential applications of CFDs in data cleaning highlight the need for further investigations of CFD discovery. (1) As remarked earlier, constant CFDs are particularly important for object identification, and thus deserve a separate treatment. One wants efficient methods to discover constant CFDs alone, without paying the price of discovering all CFDs. Indeed, as will be seen later, constant CFD discovery is often several orders of magnitude faster than general CFD discovery. (2) Levelwise algorithms [21] may not perform well on sample relations of large arity, given their inherent exponential complexity. More effective methods have to be in place to deal with datasets with a large arity. (3) A host of techniques have been developed for (non-redundant) association rule mining, and it is only natural to capitalize on these for CFD discovery. As we shall see, these techniques can not only be readily used in constant CFD discovery, but also significantly speed up general CFD discovery. To our knowledge, no previous work has considered these issues for CFD discovery.

Contributions. In light of these considerations we provide three algorithms for CFD discovery: one for discovering constant CFDs, and the other two for general CFDs.

(1) We propose a notion of minimal CFDs based on both the minimality of attributes and the minimality of patterns. Intuitively, minimal CFDs contain neither redundant attributes nor redundant patterns. Furthermore, we consider *frequent* CFDs that hold on a sample dataset r, namely, CFDs in which the pattern tuples have a support in r above a certain threshold. Frequent CFDs allow us to accommodate unreliable data with errors and noise. Our algorithms find minimal and frequent CFDs to help users identify quality cleaning rules from a possibly large set of CFDs that hold on the samples.

(2) Our first algorithm, referred to as CFDMiner, is for constant CFD discovery. We explore the connection between minimal constant CFDs and closed and free patterns. Based on this, CFDMiner finds constant CFDs by leveraging a latest mining technique proposed in [24], which mines closed itemsets and free itemsets in parallel following a depth-first search scheme.

(3) Our second algorithm, referred to as CTANE, extends TANE to discover general CFDs. It is based on an attribute-set/pattern tuple lattice, and mines CFDs at level $k + 1$ of the lattice (*i.e.,* when each set at the level consists of $k+1$ attributes) with pruning based on those at level k. CTANE discovers minimal CFDs only.

(4) Our third algorithm, referred to as FastCFD, discovers general CFDs by employing a depth-first search strategy instead of the levelwise approach. It is a nontrivial extension of FastFD mentioned above, by mining pattern tuples. A novel pruning technique is introduced by FastCFD, by leveraging constant CFDs found by CFDMiner. As opposed to CTANE, FastCFD does not take exponential time in the arity of sample data when a canonical cover of CFDs is not exponentially large.

(5) Our fifth and final contribution is an experimental study of the effectiveness and efficiency of our algorithms, based on real-life data (Wisconsin breast cancer and chess datasets from UCI) and synthetic datasets generated from data scraped from the Web. We evaluate the scalability of these methods by varying the sample size, the arity of relation schema, the active domains of attributes, and the support threshold for frequent CFDs. We find that constant CFD discovery (using CFDMiner) is often 3 orders of magnitude faster than general CFD discovery (using CTANE or FastCFD). We also find that FastCFD scales well with the arity: it is up to 3 orders of magnitude faster than CTANE when the arity is between 10 and 15, and it performs well when the arity is greater than 30; in contrast, CTANE cannot run to completion when the arity is above 17. On the other hand, CTANE is more sensitive to support threshold and outperforms FastCFD when the threshold is large and the arity is of a moderate size. We also find that our pruning techniques via itemset mining are effective: it improves the performance of FastCFD by 5-10 folds and makes FastCFD scale well with the sample size. These results provide a guideline for when to use CFDMiner, CTANE or FastCFD in different applications.

These algorithms provide a set of promising tools to help reduce manual effort in the design of data-quality rules, for users to choose for different applications. They help make CFD-based cleaning a practical data quality tool.

Organization. Section 2 defines minimal and frequent CFDs, and states the discovery problem. We present CFDMiner, CTANE and FastCFD in Sections 3, 4 and 5, respectively. The experimental results are given in Section 6, followed by related work in Section 7 and topics for future work in Section 8.

2 CFDS AND CFD DISCOVERY

We first review the definition of CFDs [1]. We then formalize the notions of minimal CFDs and frequent CFDs. Finally, we state the discovery problem for CFDs and show how the discovered CFDs are used to generate so-called *tableau* CFD*s*.

2.1 Conditional Functional Dependencies

Consider a relation schema R defined over a fixed set of attributes, denoted by $\mathtt{attr}(R)$. For each attribute $A \in \mathtt{attr}(R)$, we use $\mathbf{dom}(A)$ to denote its domain.

CFDs. A *conditional functional dependency* (CFD) φ over R is a pair $(X \to A,\ t_p)$, where (1) X is a set of attributes in $\mathtt{attr}(R)$, and A is a single attribute in $\mathtt{attr}(R)$, (2) $X \to A$ is a standard FD, referred to as the FD *embedded in* φ; and (3) t_p is a *pattern tuple* with attributes in X and A, where for each B in $X \cup \{A\}$, $t_p[B]$ is either a constant 'a' in $\mathbf{dom}(B)$, or an unnamed variable '_' that draws values from $\mathbf{dom}(B)$.

We denote X as $\mathrm{LHS}(\varphi)$ and A as $\mathrm{RHS}(\varphi)$. We separate the X and A attributes in a pattern tuple with ' $\|$ '.

Standard FDs are a special case of CFDs. Indeed, an FD $X \to A$ can be expressed as a CFD $(X \to A,\ t_p)$, where $t_p[B] = \_$ for each B in $X \cup \{A\}$.

Example 2: The FD f_1 of Example 1 can be expressed as a CFD ([CC, AC] $\to$ CT, (_, _ $\|$ _)); similarly for f_2. All of f_1, f_2 and ϕ_0–ϕ_3 are CFDs defined over schema cust. For ϕ_0, for example, $\mathrm{LHS}(\phi_0)$ is [CC, ZIP] and $\mathrm{RHS}(\phi_0)$ is STR. □

Semantics. To give the semantics of CFDs, we define an order $\leq$ on constants and the unnamed variable '_': $\eta_1 \leq \eta_2$ if either $\eta_1 = \eta_2$, or η_1 is a constant a and η_2 is '_'.

The order $\leq$ naturally extends to tuples, *e.g.*, (44, "EH4 1DT", "EDI") $\leq$ (44, _, _) but (01, 07974, "Tree Ave.") $\not\leq$ (44, _, _). We say that a tuple t_1 *matches* t_2 if $t_1 \leq t_2$. We write $t_1 \ll t_2$ if $t_1 \leq t_2$ but $t_2 \not\leq t_1$, *i.e.,* when t_2 is "more general" than t_1. For instance, (44, "EH4 1DT", "EDI") $\ll$ (44, _, _).

An instance r of R *satisfies* the CFD φ (or φ *holds* on r), denoted by $r \models \varphi$, iff for *each pair* of tuples t_1, t_2 in r, if $t_1[X] = t_2[X] \leq t_p[X]$ then $t_1[A] = t_2[A] \leq t_p[A]$.

Intuitively, φ is a constraint defined on the set $r_\varphi = \{t \mid t \in r, t[X] \leq t_p[X]\}$ such that for any $t_1, t_2 \in r_\varphi$, if $t_1[X] = t_2[X]$, then (a) $t_1[A] = t_2[A]$, and (b) $t_1[A] \leq t_p[A]$. Here (a) enforces the semantics of the embedded FD on the set r_φ, and (b) assures the binding between *constants* in $t_p[A]$ and *constants* in $t_1[A]$. That is, φ constrains *the subset* r_φ of r identified by $t_p[X]$, rather than the *entire* instance r.

Example 3: The instance r_0 of Fig. 1 satisfies CFDs f_1, f_2 and ϕ_0–ϕ_3 of Example 1. It does *not* satisfy the CFD ψ = ([CC, ZIP] $\to$ STR, (_, _ $\|$ _)). Indeed, t_1 and t_4 *violate* ψ since t_1[CC, ZIP] = t_4[CC,ZIP] $\leq$ (_, _), but t_1[STR] $\neq$ t_4[STR]. Nor does r satisfy ψ' = (AC $\to$ CT, (131 $\|$ EDI)) since t_8 violates ψ': t_8[AC] $\leq$ (131) but t_8[CT] $\not\leq$ (EDI). From this one can see that while *two* tuples are needed to violate an FD, CFDs can be violated by a *single* tuple. □

We say that an instance r of R *satisfies* a set Σ of CFDs over R, denoted by $r \models \Sigma$, if $r \models \varphi$ for *each* CFD $\varphi \in \Sigma$.

For two sets Σ and Σ' of CFDs defined over the same schema R, we say that Σ is *equivalent to* Σ', denoted by $\Sigma \equiv \Sigma'$, iff for any instance r of R, $r \models \Sigma$ iff $r \models \Sigma'$.

Remark. CFDs can also be defined as $(X \to Y,\ t_p)$, where Y is a set of attributes and $X \to Y$ is an FD. As in the case of FDs, such a CFD is equivalent to a set of CFDs with a single attribute in their RHS. Thus in the sequel we focus on CFDs with their RHS consisting of a single attribute.

Classification of CFDs. A CFD $(X \to A,\ t_p)$ is called a *constant* CFD if its pattern tuple t_p consists of constants only, *i.e.,* $t_p[A]$ is a constant and for all $B \in X$, $t_p[B]$ is a constant. It is called a *variable* CFD if $t_p[A] = \_$, *i.e.,* the RHS of its pattern tuple is the unnamed variable '_'.

Example 4: Among the CFDs given in Example 1, f_1, f_2, ϕ_0 are variable CFDs, while ϕ_1, ϕ_2, ϕ_3 are constant CFDs. $\square$

It has been shown in [1] that any set Σ of CFDs over a schema R can be represented by a set Σ_c of constant CFDs and a set Σ_v of variable CFDs, such that $\Sigma \equiv \Sigma_c \cup \Sigma_v$. In particular, for a CFD $\phi = (X \rightarrow A,\ t_p)$, if $t_p[A]$ is a constant a, then there is an equivalent CFD $\phi' = (X' \rightarrow A,\ (t_p[X'] \parallel a))$, where X' consists of all attributes $B \in X$ such that $t_p[B]$ is a constant. That is, when $t_p[A]$ is a constant, we can safely drop all attributes B in the LHS of ϕ with $t_p[B]$ = '_'.

Lemma 1:*[1] For any set Σ of* CFD*s over a schema R, there exist a set Σ_c of constant* CFD*s and a set Σ_v of variable* CFD*s over R, such that Σ is equivalent to $\Sigma_c \cup \Sigma_v$.* $\square$

2.2 The Discovery Problem for CFDs

Given a sample relation r of a schema R, an algorithm for CFD discovery aims to find CFDs defined over R that hold on r. Obviously it is not a good idea to return the set of all CFDs that hold on r, since the set contains trivial and redundant CFDs and is unnecessarily large. Thus we want to find a canonical cover, *i.e.,* a non-redundant set consisting of *minimal* CFDs only, from which all CFDs on r can be derived via implication analysis. Moreover, real-life data is often dirty, containing errors and noise. To exclude CFDs that match errors and noise only, we consider *frequent* CFDs, which have a pattern tuple with support in r above a threshold.

Below we first formalize the notions of minimal CFDs and frequent CFDs. We then state the discovery problem for CFDs.

Minimal CFDs. A CFD $\varphi = (X \rightarrow A,\ t_p)$ over R is said to be *trivial* if $A \in X$. If φ is trivial, then either it is satisfied by all instances of R (*e.g.,* when $t_p[A_L] = t_p[A_R]$), or it is satisfied by none of the instances in which there is a tuple t such that $t[X] \leq t_p[X]$ (*e.g.,* if $t_p[A_L]$ and $t_p[A_R]$ are distinct constants). In the sequel we consider nontrivial CFDs only.

A constant CFD $(X \rightarrow A,\ (t_p \parallel a))$ is said to be *left-reduced* on r if for any $Y \subsetneq X$, $r \not\models (Y \rightarrow A, (t_p[Y] \parallel a))$.

A variable CFD $(X \rightarrow A,\ (t_p \parallel \_))$ is *left-reduced* on r if (1) $r \not\models (Y \rightarrow A, (t_p[Y] \parallel \_))$ for any proper subset $Y \subsetneq X$, and (2) $r \not\models (X \rightarrow A, (t'_p[X] \parallel \_))$ for any t'_p with $t_p \ll t'_p$.

Intuitively, these assure the following: (1) none of its LHS attributes can be removed, *i.e.,* the minimality of attributes, and (2) none of the constants in its LHS pattern can be "upgraded" to '_', *i.e.,* the pattern $t_p[X]$ is "most general", or in other words, the minimality of patterns.

A *minimal* CFD φ *on* r is a nontrivial, left-reduced CFD such that $r \models \varphi$. Intuitively, a minimal CFD is non-redundant.

Example 5: On the sample r_0 of Fig. 1, ϕ_2 of Example 1 is a minimal constant CFDs, and f_1, f_2 and ϕ_0 are minimal variable CFDs. However, ϕ_3 is not minimal: if we drop CC from LHS(ϕ_3), r_0 still satisfies (AC $\rightarrow$ CT, (212 $\parallel$ NYC)) since there is only one tuple (t_3) with AC = 212 in r_0. Similarly, ϕ_1 is not minimal since CC can be dropped.

Consider $f_1^1 = (f_1, (01, \_ \parallel \_))$, $f_1^2 = (f_1, (44, \_ \parallel \_))$, $f_1^3 = (f_1, (\_, 908 \parallel \_))$, $f_1^4 = (f_1, (\_, 212 \parallel \_))$, and $f_1^5 = (f_1, (\_, 131 \parallel \_))$. While these CFDs hold on r_0, they are not minimal CFDs, since they do not satisfy requirement (2) for left-reduced variable CFDs. Indeed, $(f_1, (\_, \_ \parallel \_))$ is a minimal CFD on r_0 with a pattern more general than any of f_1^i for $i \in [1, 5]$; in other words, these f_1^i's are *redundant*. $\square$

Frequent CFDs. The *support* of a CFD $\varphi = (X \rightarrow A,\ t_p)$ in r, denoted by $\sup(\varphi, r)$, is defined to be the set of tuples t in r such that $t[X] \leq t_p[X]$ and $t[A] \leq t_p[A]$, *i.e.,* tuples that match the pattern of φ. For a natural number $k \geq 1$, a CFD φ is said to be *k-frequent* in r if $\sup(\varphi, r) \geq k$. For instance, ϕ_1 and ϕ_2 of Example 1 are 3-frequent and 2-frequent, respectively. Moreover, f_1 and f_2 are 8-frequent.

It should be mentioned that the notion of frequent CFDs is quite different from the notion of approximate FDs [13], [18]. An approximate FD ψ on a relation r is an FD that "almost" holds on r, *i.e.,* there exists a subset $r' \subseteq r$ such that $r' \models \psi$ and the error $|r \setminus r'|/|r|$ is less than a predefined bound. It is not necessary that $r \models \psi$. In contrast, a k-frequent CFD φ in r is a CFD that must hold on r, *i.e.,* $r \models \varphi$, and moreover, there must be sufficiently many (at least k) witness tuples in r that match the pattern tuple of φ.

Problem statement. A *canonical cover* of CFDs on r *w.r.t.* k is a set Σ of minimal, k-frequent CFDs in r, such that Σ is equivalent to the set of all k-frequent CFDs that hold on r.

Given an instance r of a relation schema R and a support threshold k, the *discovery problem* for CFDs is to find a canonical cover of CFDs on r *w.r.t.* k. Intuitively, a canonical cover consists of non-redundant frequent CFDs on r, from which all frequent CFDs that hold on r can be inferred.

2.3 Discovering CFDs with pattern tableaux

So far, we have considered CFDs of the form $\varphi = (X \rightarrow A, t_p)$. In [1], however, CFDs were allowed to have *multiple* pattern tuples. More specifically, a *tableau* CFD is of the form $\varphi = (X \rightarrow A, T_p)$ where T_p is a *pattern tableau* consisting of a finite number of pattern tuples with attributes in X and A. An instance r of R is said to satisfy φ if r satisfies *every* CFD $\varphi_{t_p} = (X \rightarrow A, t_p)$ with $t_p \in T_p$. It is easily verified (see [1]) that a tableau CFD $\varphi = (X \rightarrow A, T_p)$ is equivalent to the set of CFDs $\{\varphi_{t_p} \mid t_p \in T_p\}$. Motivated by this equivalence, we define the support of $\varphi = (X \rightarrow A, T_p)$ in r, denoted by $\sup(\varphi, r)$, as $\min_{t_p \in T_p} \sup(\varphi_{t_p}, r)$. Hence, the discovery of k-frequent tableau CFDs reduces to the problem of discovering k-frequent CFDs. We remark that an alternative definition of support for tableau CFDs is considered in [32].

Furthermore, the notion of minimality extends to tableau CFDs: A CFD $\varphi = (X \rightarrow A, T_p)$ is minimal on r if (1) $r \not\models (Y \rightarrow A, T_p)$ for any proper subset $Y \subsetneq X$; and (2) the pattern tableau is maximal (it cannot be extended with more pattern tuples) without violating the condition that for any two pattern tuples s_p and t_p, if $s_p[X] \leqslant t_p[X]$ then $t_p[A] \nleqslant s_p[A]$.

It is readily verified that k-frequent minimal tableau CFDs can be obtained from k-frequent minimal (single pattern tuple) CFDs. We therefore focus on the latter in this paper.

3 Discovering Constant CFDs

In this section we present CFDMiner, our algorithm for constant CFD discovery. Given an instance r of R and a support threshold k, CFDMiner finds a canonical cover of k-frequent minimal constant CFDs of the form $(X \to A, (t_p \parallel a))$.

Our algorithm is based on the connection between left-reduced constant CFDs and *free* and *closed itemsets*. A similar relationship was established for so-called non-redundant association rules [23]. In that context, constant CFDs coincide with association rules that have 100% confidence and have a single attribute in their antecedent. Non-redundant association rules, however, do not precisely correspond to left-reduced constant CFDs. Indeed, non-redundancy is only defined for association rules with the same support. In contrast, left-reducedness requires the comparison of constant CFDs with different supports. Finally, whereas [23] provides algorithms based on closed sets, our algorithm is based on both closed and free sets. Hence the need to revisit the relationship between minimal constant CFDs and itemset mining.

To make the relationship more precise, we first recall the notions of free and closed itemsets [23].

Free and closed itemsets. An itemset is a pair (X, t_p), where $X \subseteq \mathtt{attr}(R)$ and t_p is a constant pattern over X.

Given an instance r of the schema R, the *support* of (X, t_p) in r, denoted by $\mathsf{supp}(X, t_p, r)$, is defined as the set of tuples in r that match with t_p on the X-attributes.

We say that (Y, s_p) is *more general* than (X, t_p), denoted by $(X, t_p) \preceq (Y, s_p)$, if $Y \subseteq X$ and $s_p = t_p[Y]$. Furthermore, (Y, s_p) is said to be *strictly more general than* (X, t_p), denoted by $(X, t_p) \prec (Y, s_p)$, if $Y \subset X$ and $t_p[Y] = s_p$. Clearly, if $(X, t_p) \preceq (Y, s_p)$ then $\mathsf{supp}(X, t_p, r) \subseteq \mathsf{supp}(Y, s_p, r)$.

An itemset (X, t_p) is called *closed* in r if there is no itemset (Y, s_p) such that $(Y, s_p) \preceq (X, t_p)$ with $\mathsf{supp}(Y, s_p, r) = \mathsf{supp}(X, t_p, r)$. Intuitively, a closed itemset (X, t_p) cannot be extended without decreasing its support. For an itemset (X, t_p), we denote by $\mathsf{clo}(X, t_p)$ the unique closed itemset that extends (X, t_p) and has the same support in r as (X, t_p).

Similarly, an itemset (X, t_p) is called *free* in r if there exists no itemset (Y, s_p) such that $(X, t_p) \preceq (Y, s_p)$ for which $\mathsf{supp}(Y, s_p, r) = \mathsf{supp}(X, t_p, r)$. Intuitively, a free itemset (X, t_p) cannot be generalized without increasing its support.

For a natural number $k \geq 1$, a closed (resp. free) itemset (X, t_p) is called *k-frequent* if $|\mathsf{supp}(X, t_p, r)| \geq k$.

Example 6: Figure 2 shows the closed sets in the cust relation (see Fig. 1) that contain $(\mathsf{CT}, (\text{MH}))$. It also shows the corresponding free sets (closed sets are enclosed in a rectangle). To simplify the figure, we do not show the attribute names in the itemsets, but we show the size of the support of the itemsets. For example, ([CC, AC, CT, ZIP], (01, 908, MH, 07974)) is a closed itemset with support equal to 3. This itemset has two free patterns, ([CC, AC], (01, 908)) and ([ZIP],(07974)), both having support = 3 as well. $\square$

The connection between k-frequent free and closed itemsets and k-frequent left-reduced constant CFDs is as follows.

Proposition 1: *For an instance r of R and any k-frequent left-reduced constant* CFD $\varphi = (X \to A, (t_p \parallel a))$, $r \models \varphi$ *iff (i) the itemset (X, t_p) is free, k-frequent and it does not contain (A, a); (ii) $\mathsf{clo}(X, t_p) \preceq (A, a)$; and (iii) (X, t_p) does not contain a smaller free set (Y, s_p) with this property, i.e., there exists no (Y, s_p) such that $(X, t_p) \preceq (Y, s_p)$, $Y \subsetneq X$, and $\mathsf{clo}(Y, s_p) \preceq (A, a)$.* $\square$

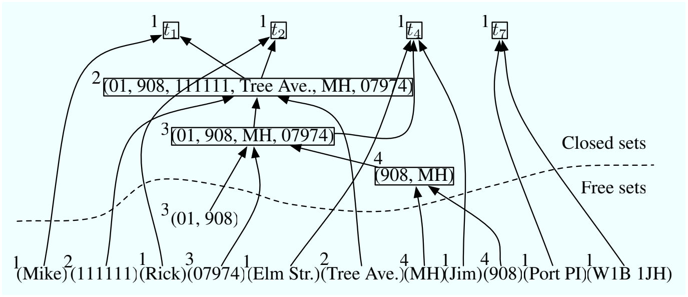

Fig. 2. Closed sets in a cust relation that contains $(\mathsf{CT}, (\text{MH}))$ and their corresponding free sets.

Proof: Suppose that $r \models \varphi$, where $\varphi = (R : X \to A, (t_p \parallel a))$ and it is left-reduced and k-frequent. Then from the semantics of CFDs it follows that $\mathsf{supp}(X, t_p) = \mathsf{supp}([X, A], (t_p, a))$ and $|\mathsf{supp}([X, A], (t_p, a))| \geqslant k$. Moreover, the minimality of φ ensures that there is no (strict) subset Y of X such that $\mathsf{supp}(Y, t_p[Y])) = \mathsf{supp}([Y, A], (t_p[Y], a))$. In other words, (X, t_p) is a k-frequent itemset that is free and is minimal among all free itemsets $(Y, t_p[Y])$ that satisfy $\mathsf{supp}(Y, t_p[Y])) = \mathsf{supp}([Y, A], (t_p[Y], a))$. Hence (X, t_p) is the minimal k-frequent itemsets among all such itemsets.

Conversely, let (X, t_p) be a k-frequent free itemset such that $\mathsf{clo}(X, t_p)$ contains (A, a) and (X, t_p) is minimal among all such itemsets. Then clearly $r \models \varphi$, where $\varphi = (R : X \to A, (t_p \parallel a))$; in addition, the same reasoning as above shows that it is a k-frequent minimal CFD. $\square$

Example 7: From proposition 1 and the closed and free itemsets shown in Fig. 2, it follows that ϕ_1: ([CC, AC] $\to$ CT, (01, 908 $\parallel$ MH)) of Example 1 is a 3-frequent constant CFD that holds on the cust relation. Indeed, it is obtained from the closed pattern ([CC, AC, CT, ZIP], (01, 908, MH, 07974)), where the free pattern ([CC, AC], (01, 908)) is taken as the LHS of the constant CFD. Figure 2, however, shows that this LHS contains a smaller free set (AC, (908)) whose closed set ([AC, CT], (908, MH)) contains (CT, (MH)). Hence ϕ_1 is not left-reduced. It is easily verified that (AC $\to$ CT, (908 $\parallel$ MH)) is a 4-frequent left-reduced constant CFD on cust. Similarly ϕ_2 and ϕ_3 of Example 1 can be obtained (although one has to consider closed patterns that contain (CT, (EDI)) for ϕ_2). $\square$

CFDMiner. Proposition 1 forms the basis for our constant CFD discovery algorithm. Suppose that for a given instance r and a support threshold k, we have all k-frequent closed sets and their corresponding k-frequent free sets at our disposal. Algorithm CFDMiner then finds k-frequent left-reduced constant CFDs from these sets. As mentioned in Section 1, there have been various algorithms that provide these sets [25]. We opt for the GCGROWTH algorithm of [25] because it, in contrast to other algorithms, simultaneously discovers closed sets and their free sets. Due to space limitations we omit the details of algorithm GCGROWTH; we refer the reader to [25] for more details. For our purposes, it is sufficient to know that GCGROWTH returns a mapping C2F that associates with each k-frequent closed itemset its set of k-frequent free itemsets.

Given this mapping, CFDMiner works as follows:

(1) For each k-frequent closed itemset (X, t_p) we add its free sets, as given by C2F, to a hash table H.

(2) For each closed itemset (X, t_p), we associate with each of its free itemsets (Y, s_p) the itemset $\text{RHS}(Y, s_p) = (X \setminus Y, t_p[X \setminus Y])$. That is, we associate with each free set the candidate RHS attributes in their corresponding constant CFDs.

During this process, an ordered list L of all k-frequent free itemsets is constructed as well. Itemsets in this list are sorted in ascending order *w.r.t.* their sizes.

(3) For each free itemset (Y, s_p) in the list L, CFDMiner does the following:

(a) For each subset $Y' \subsetneq Y$ such that $(Y', s_p[Y']) \in$ L, it replaces $\text{RHS}(Y, s_p)$ with $\text{RHS}(Y, s_p) \cap \text{RHS}(Y', s_p[Y'])$. Indeed, Proposition 1 implies that only those elements in $\text{RHS}(Y, s_p)$ can lead to a left-reduced constant CFD that are not already included in some $\text{RHS}(Y', s_p[Y'])$ of one of its sub-itemsets. It is important to remark that the subset checking can be done efficiently by leveraging the hash-table H.

(b) After all subsets of (Y, s_p) are checked, CFDMiner outputs k-frequent constant CFDs $(Y \to A, (s_p \parallel a))$ for all $(A, a) \in \text{RHS}(Y, s_p)$.

As will be verified in Section 6, this yields an efficient algorithm for discovering constant CFDs.

4 CTANE: A Levelwise Algorithm

We next present CTANE, a levelwise algorithm for discovering minimal, k-frequent (variable *and* constant) CFDs. It is an extension of algorithm TANE [13] for discovering FDs.

CTANE mines CFDs by traversing an attribute-set/pattern lattice $\mathcal{L}$ in a levelwise way. More precisely, the lattice $\mathcal{L}$ consists of elements of the form (X, t_p), where $X \subseteq \mathbf{attr}(R)$ and t_p is pattern tuple over X. In contrast to the itemsets in Section 3, the patterns now consist of both constants and unnamed variables (_). We say that (Y, s_p) is *more general* than (X, t_p) if $Y \subseteq X$ and $t_p[Y] \ll s_p$. This relationship defines the lattice structure on the attribute-set/pattern pairs.

We first present CTANE for mining 1-frequent minimal CFDs. We then describe how to modify CTANE to discover k-frequent minimal CFDs for a support threshold k.

CTANE starts from singleton sets (A, α) for $A \in \mathbf{attr}(R)$ and $\alpha \in \mathbf{dom}(A) \cup \{\_\}$. It then proceeds to larger attribute-set/pattern levels in $\mathcal{L}$. When it inspects (X, s_p), it checks CFDs $(X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel s_p[A]))$, where $A \in X$. This guarantees that only non-trivial CFDs are considered. Furthermore, CTANE maintains for each considered element (X, s_p) a set, denoted by $\mathcal{C}^+(X, s_p)$, to determine whether CFD $(X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel s_p[A]))$ is minimal. The set $\mathcal{C}^+(X, s_p)$, as will be elaborated below, can be maintained during the levelwise traversal. Apart from testing for minimality, $\mathcal{C}^+(X, s_p)$ also provides an effective pruning strategy, making the levelwise approach feasible in practice.

Pruning strategy. To efficiently discover CFDs, we first extend TANE's pruning strategy. For each element (X, s_p) in $\mathcal{L}$, we provide a set $\mathcal{C}^+(X, s_p)$ that consists of elements $(A, c_A) \in \mathbf{attr}(R) \times \{\mathbf{dom}(A) \cup \{\_\}\}$, satisfying conditions:

1) if $A \in X$, then $c_A = s_p[A]$;
2) $r \not\models (X \setminus \{A, B\} \to B, (s_p[X \setminus \{A, B\}] \parallel s_p[B]))$ for all $B \in X$; and
3) for all $B \in X \setminus \{A\}$, $r \not\models (X \setminus \{A\} \to A, (s_p^B \parallel c_A))$, where $s_p^B[C] = s_p[C]$ for all $C \neq B$ and $s_p^B[B] = \_$.

Intuitively, condition 1 prevents the creation of inconsistent CFDs; condition 2 ensures that the LHS cannot be reduced; and condition 3 ensures that the pattern tuple is most general.

The following is easily verified:

Lemma 2: *Let $X \subseteq \mathbf{attr}(R)$, s_p be a pattern over X, $A \in X$ and assume that $r \models \varphi = (X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel s_p[A]))$. Then φ is minimal iff for all $B \in X$ we have that $(A, s_p[A]) \in \mathcal{C}^+(X \setminus \{B\}, s_p[X \setminus \{B\}])$.* □

In terms of pruning, Lemma 2 says that we do not need to consider any element (X, s_p) of $\mathcal{L}$ for which $\mathcal{C}^+(X, s_p) = \emptyset$. Moreover, if $\mathcal{C}^+(X, s_p) = \emptyset$ then also $\mathcal{C}^+(Y, t_p) = \emptyset$ for any (Y, s_p) that contains (X, t_p) in the lattice. Therefore, the emptiness of $\mathcal{C}^+(X, s_p)$ potentially prunes away a large part of elements in $\mathcal{L}$ that otherwise need to be considered by CTANE.

Proof: Recall that we assume that $r \models (X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel c_A))$ for some $A \in X$.

Assume that $\varphi = (X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel c_A))$ is not minimal. We distinguish between the following two cases.

First, assume that there exists $C \in X, C \neq A$ such that $r \models (X \setminus \{A, C\} \to A, (s_p[X \setminus \{A, C\}] \parallel c_A))$. Since $c_A = s_p[A]$ (by condition 1 and the fact that $A \in X$), this implies that $r \models (X \setminus \{A, C\} \to A, (s_p[X \setminus \{A, C\}] \parallel s_p[A]))$. Therefore, $(A, c_A) \notin C^+(X \setminus \{C\}, t_p[X \setminus \{C\}])$ because condition 2 is not satisfied for the choice of $B = A$.

Second, assume that there exists $X_i \in X \setminus \{A\}$ such that $r \models (X \setminus \{A\} \to A, (s_p^i[A], c_A))$. Then clearly, $(A, c_A) \notin \mathcal{C}^+(X \setminus \{A\}, s_p[X \setminus \{A\}])$ because condition 3 is not satisfied. As a consequence, if φ is not minimal then $(A, c_A) \notin \bigcap_{B \in X} \mathcal{C}^+(X \setminus \{B\}, s_p[X \setminus \{B\}])$.

Conversely, suppose that $(A, c_A) \notin \bigcap_{B \in X} \mathcal{C}^+(X \setminus \{B\}, s_p[X \setminus \{B\}])$. We need to show that $\varphi = (X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel c_A))$ is not minimal. Let B be such that $(A, c_A) \notin \mathcal{C}^+(X \setminus \{B\}, s_p[X \setminus \{B\}])$. We distinguish between the following two cases.

First assume that condition 2 is violated, *i.e.,* there exists $C \in X \setminus \{B\}$ such that $r \models (X \setminus \{A, B, C\} \to C, (s_p[X \setminus \{A, B\}] \parallel s_p[C]))$. If $C = A$, then $B \neq A$ and therefore, $r \models (X \setminus \{A, B\} \to A, (s_p[X \setminus \{A, B\}] \parallel s_p[A]))$. However, given that $r \models (X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel c_A))$ and $c_A = s_p[A]$, this implies that $r \models (X \setminus \{A, B\} \to A, (s_p[X \setminus \{A, B\}] \parallel c_A))$. Hence φ is not minimal. If $C \neq A$, then $r \models (X \setminus \{A, C\} \to C, (s_p[X \setminus \{A, B\}] \parallel s_p[C]))$. By transitivity, we therefore also have that $r \models (X \setminus \{A, C\} \to A, (s_p[X \setminus \{A, B\}] \parallel c_A))$, from which the non-minimality of φ follows.

Second, assume that condition 3 is violated. Then there are two cases to consider. (a) If $(A, c_A) \notin \mathcal{C}^+(X \setminus \{A\}, s_p[X \setminus \{A\}])$ then this implies directly that φ is not minimal. (b) If $(A, c_A) \notin \mathcal{C}^+(X \setminus \{B\}, s_p[X \setminus \{B\}])$, for $B \neq A$, then along the same lines as above it can be shown that one can remove an extra attribute from $X \setminus \{A\}$ in φ. □

Algorithm CTANE. We are now ready to present the algorithm. We denote by L_ℓ a collection of elements (X, s_p) in $\mathcal{L}$ of size ℓ, *i.e.,* $|X| = \ell$. We assume that L_ℓ is ordered such that (X, s_p) appears before (Y, t_p) if $X = Y$ and $t_p \ll s_p$. Initially, $L_1 = \{(A, \_) \mid A \in \mathbf{attr}(R)\} \cup \{(A, a_1) \mid a_1 \in \pi_A(r), A \in \mathbf{attr}(R)\}$, $\mathcal{C}^+(\emptyset) = L_1$ and $\ell = 1$. We then execute the following steps as long as L_ℓ is non-empty:

1. We compute candidate RHS for minimal CFDs with their LHS in L_ℓ. That is, for each $(X, s_p) \in L_\ell$ we compute

$$\mathcal{C}^+(X, s_p) = \bigcap_{B \in X} \mathcal{C}^+(X \setminus \{B\}, s_p[X \setminus \{B\}]);$$

2. For each $(X, s_p) \in L_\ell$ we look for valid CFDs; *i.e.,* for each $A \in X$, $(A, c_A) \in \mathcal{C}^+(X, s_p)$ we do the following:
 (a) check whether $r \models \varphi$, where
$$\varphi = (X \setminus \{A\} \to A, (s_p[X \setminus \{A\}] \parallel c_A));$$
 (b) if $r \models \varphi$ then output φ. Indeed, if φ holds on r then by Lemma 2 and Step 1, φ is indeed a minimal CFD;
 (c) if $r \models \varphi$ then for all $(X, u_p) \in L_\ell$ such that $u_p[A] = c_A$ and $u_p[X \setminus \{A\}] \ll s_p[X \setminus \{A\}]$, update $\mathcal{C}^+(X, u_p)$ by removing (A, c_A) and (B, c_B) for all $B \in \mathbf{attr}(R) \setminus X$;

3. Next, we prune L_ℓ. That is, for each $(X, s_p) \in L_\ell$ we remove (X, s_p) from L_ℓ provided that $\mathcal{C}^+(X, s_p) = \emptyset$;

4. Finally, we generate $L_{\ell+1}$ as follows:
 (a) initially $L_{\ell+1} = \emptyset$;
 (b) for each pair of distinct $(X, s_p), (Y, t_p) \in L_\ell$ that agree on the first $\ell - 1$ attributes:
  (i) let $Z = X \cup Y$ and $u_p = (s_p, t_p[Y_n])$; here Y_n denotes the last attribute in Y;
  (ii) if there is a tuple in the projection $\pi_Z(r)$ that matches u_p then continue with (Z, u_p);
  (iii) if for all $A \in Z$, $(Z \setminus \{A\}, u_p[Z \setminus \{A\}]) \in L_\ell$, then add (Z, u_p) to $L_{\ell+1}$;
 (c) set $\ell = \ell + 1$.

Before we prove the correctness of the algorithm, we first extend the algorithm to find k-frequent CFDs, and illustrate how it works with an example.

CTANE for finding k-frequent CFDs. CTANE can be easily modified such that it only discovers k-frequent minimal CFDs. First we observe the following. Let $\varphi = (X \to A, (t_p, c_A))$ be a CFD that holds on r. We denote by (X^c, t_p^c) the itemset consisting of the constant part of (X, t_p). Then φ is k-frequent iff $\mathsf{supp}(X^c, t_p^c, r) \geq k$ when $X \neq \emptyset$ and $|r| \geq k$.

This tells us that for any reasonable choice of k (*i.e.,* smaller than the size of r), we only need to restrict the elements $(X, s_p) \in L_\ell$ to those for which (X^c, s_p^c) is a k-frequent itemset. This can be achieved by (1) starting with $L_1 = \{(A, \_) \mid A \in \mathbf{attr}(R)\} \cup \{(A, a_1) \mid \mathsf{supp}(A, a_1, r) \geq k, A \in \mathbf{attr}(R)\}$; and (2) replacing Step 4.b(ii) in CTANE by a step that only considers (Z, u_p) if $\mathsf{supp}(Z^c, u_p^c, r) \geq k$. Both modifications yield more pruning, and thus improve the efficiency of CTANE when finding k-frequent CFDs.

Example 8: Consider again the cust relation of Fig 1. We give a partial run of algorithm CTANE involving only attributes CC, AC, ZIP and STR. Assume a support threshold $k \geq 3$.

We show in Fig. 3 the first two levels of lattice $\mathcal{L}$ and the third level corresponding to attributes [CC, AC, ZIP]. In particular, for each element (X, s_p) inspected by CTANE we list the attribute set X together with the list of possible patterns, ranked *w.r.t.* the number of '_' in them.

We highlight certain points during the execution of CTANE: A, B, C, D, E, F reached in this order, as indicated in Fig. 3.

(A) Initially L_1 consists of all single attribute/value pairs that appear at least k times, and each attribute occurs together with an unnamed variable. Note that k limits the number of values dramatically for, *e.g.,* the STR attribute. At this point, all sets $C^+(A, c_A)$ contain (A, c_A). Since r does not satisfy any CFD with an empty LHS, none of the C^+-sets is updated in Step 2. Similarly, none of the sets is removed from L_1 in Step 3.

(B) In Step 4, CTANE pairs attributes together and creates consistent patterns. Note that for (CC, AC) the constant 44 does not appear anywhere (while it did at the lower level). This is because $k = 3$.

(C) For the gray shaded patterns, Step 2 finds valid CFDs: (ZIP $\to$ CC, (07974 $\parallel$ _)), (ZIP $\to$ CC, (07974 $\parallel$ 01)), (ZIP $\to$ AC, (07974 $\parallel$ _)), (ZIP $\to$ AC, (07974 $\parallel$ 908)), and (STR $\to$ ZIP, (_ $\parallel$ _)). This implies that, *e.g.,* C^+([CC, ZIP], (_,07974)) and C^+([AC, ZIP], (_,07974)) are updated in Step 2 by removing (CC, _) and (AC, _), respectively.

(D) Step 4 now creates triples of attributes. We only show the patterns for (CC, AC, ZIP). In Step 2, CTANE finds the CFD ([CC, AC] $\to$ ZIP, (_, _ $\parallel$ _)).

(E) As a result, CTANE updates the C^+-sets in Step 2.c, not only of the current patten but also of those with a more specific pattern on the LHS-attributes. That is, (ZIP, _) is removed from the C^+-set from the first three patterns. This ensures that CFD to be generated later only have the most general LHS-pattern.

(F) Finally, in Step 1 of CTANE, the C^+ set of the pattern tuple (_, _,07974) is computed. However, recall that both C^+([CC, ZIP], (_,07974)) and C^+([AC, ZIP], (_,07974)) have been updated. As a result, neither (CC, _) nor (AC, _) will be included in the C^+-set of (_, _,07974). This illustrates that the only chance of finding an minimal CFD in this case is to test ([AC, CC] $\to$ ZIP, (_, _ $\parallel$ 07974)), which in this case does not hold on r. However, this shows that the C^+-sets indeed reduce the possible RHS for candidate minimal CFDs. □

Correctness. As for algorithm TANE, Lemma 2 ensures that Steps 1 and 2.a of algorithm CTANE correctly generates minimal CFDs. Further, it is easily verified that Steps 1 and 2.c of CTANE correctly update $\mathcal{C}^+(X, s_p)$:

Lemma 3: *Suppose that for all $(Y, t_p) \in L_\ell$, $\mathcal{C}^+(Y, t_p)$ is correctly computed. Then steps 1 and 2.c of CTANE correctly compute $\mathcal{C}^+(X, s_p)$ for all $(X, s_p) \in L_{\ell+1}$.* □

Proof: Suppose that (B, c_B) is not in $\mathcal{C}^+(X, s_p)$. We show

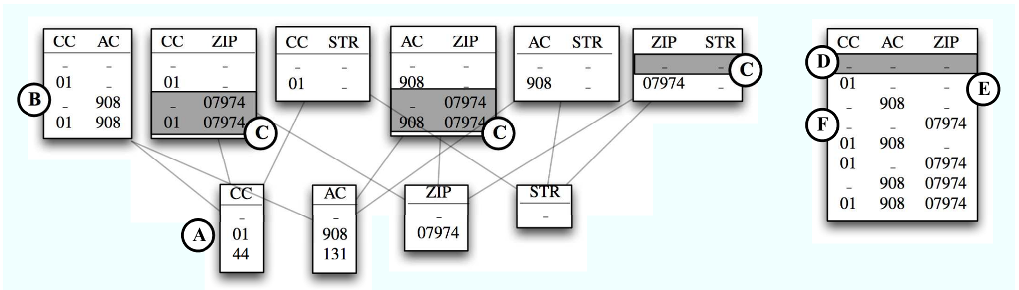

Fig. 3. Partial execution of algorithm CTANE. Explanation of circled A, B, C, D, E and F are provided in Example 8.

that (B, c_B) is indeed removed by CTANE. Clearly $(B, c_B) \notin \mathcal{C}^+(X, s_p)$ if (a) either there exists $C \in X$ such that $r \models (X \setminus \{B, C\} \rightarrow C, (s_p[X \setminus \{B, C\}] \parallel s_p[C]))$; or (b) there exists $X_i \in X \backslash B$ such that $r \models (X \backslash \{B\} \rightarrow B, (s_p^i[X \backslash \{B\}] \parallel c_B))$.

For case (a), we distinguish between the following cases:

(a_1) $B \in X$ and $B = C$. Then $r \models (X \setminus \{B\}] \rightarrow B, (s_p[X \setminus \{B\}] \parallel s_p[B]))$. If $(X \setminus \{B\}] \rightarrow B, (s_p[X \setminus \{B\}] \parallel s_p[B]))$ is minimal, then $(B, s_p[B])$ is removed in step 2.c from $C^+(X, s_p)$ (note that $(B, s_p[B])$ is equal to (B, c_B) by condition 1 described in the pruning strategy, since $B \in X$). Otherwise, (B, c_B) is removed in step 1.

(a_2) $B \in X$ and $B \neq C$. Then we have that $r \models (X \setminus \{B, C\}] \rightarrow C, (s_p[X \setminus \{B, C\} \parallel s_p[C]))$. In this case $(B, c_B) \notin \mathcal{C}^+(X \setminus \{B\}, s_p[X \setminus \{B\}])$ and (B, c_B) is hence removed in step 1.

(a_3) $B \notin X$, then $B \neq C$. Then $r \models (X \setminus \{C\}] \rightarrow C, (s_p[X \setminus \{C\} \parallel s_p[C]))$. In this case if $(X \setminus \{C\}] \rightarrow C, (s_p[X \setminus \{C\} \parallel s_p[C]))$ is minimal, then (B, c_B) is removed in 2.c. Otherwise it is removed in step 1.

For case (b), we proceed as follows:

(b) Let t_p be the most general pattern over $X \setminus \{B\}$ such that $r \models (X \setminus \{B\} \rightarrow B, (t_p \parallel c_B))$. Recall now that L_ℓ is ordered such that more general patterns are consider first. This implies that $C^+(X, (t_p, c_B))$ has been considered before $C^+(X, s_p)$ in step 2. Assume now that $(B, c_B) \in C^+(X, (t_p, c_B))$ or in other words, that $(X \setminus \{B\} \rightarrow B, (t_p \parallel c_B))$ is minimal. Then (B, c_B) is removed in Step 2.c from $C^+(X, (t_p, c_B))$. Moreover, Step 2.c also removes (B, c_B) from $C^+(X, u_p)$ for all patterns u_p such that $u_p[B] = c_B$ and $u_p[X \setminus \{B\}] \preceq t_p$. In particular, (B, c_B) is removed from $C^+(X, s_p)$, as desired.
Next, consider the case when $(B, c_B) \notin C^+(X, (t_p, c_B))$. Since by definition, t_p is the most general pattern for which $r \models (X \setminus \{B\} \rightarrow B, (t_p \parallel c_B))$, this implies that $(X \setminus \{B\} \rightarrow B, (t_p \parallel c_B))$ is not minimal because one can eliminate an attribute from $X \setminus \{B\}$. However, this also implies that $(B, c_B) \notin C^+(X, s_p)$ because of case (a) that we dealt with above. As a result, (B, c_B) will be removed from $C^+(X, s_p)$ as desired.

We next show the converse: if (B, c_B) is removed by CTANE then it indeed could not have been in $\mathcal{C}^+(X, s_p)$. We consider the following cases:

- If (B, c_B) is removed in step 1, then this implies that there exists C such that $(B, c_B) \notin C^+(X \setminus \{C\}, s_p[X \setminus \{C\}])$. More specifically, there are two cases.
(i) There exists $D \in X \setminus \{C\}$ such that $r \models (X \setminus \{B, C, D\} \rightarrow D, (s_p[X \setminus \{B, C, D\}] \parallel s_p[D]))$. From this it follows that $r \models (X \setminus \{B, D\} \rightarrow D, (s_p[X \setminus \{B, D\}] \parallel s_p[D]))$, and hence $(B, c_B) \notin C^+(X, s_p)$.
(ii) $r \models (X \setminus \{B, C\} \rightarrow B, (s_p^i[X \setminus \{B, C\}] \parallel c_B))$; then $r \models (X \setminus \{B\} \rightarrow B, (s_p^i[X \setminus \{B\}] \parallel c_B))$, and hence $(B, c_B) \notin C^+(X, s_p)$.
- If (B, c_B) is removed in 2.c and $B \in X$, then we have that $r \models (X \setminus \{B\} \rightarrow B, (t_p[X \setminus \{B\} \parallel c_B))$, where $s_p[X \setminus \{B\}] \preceq t_p[X \setminus \{B\}]$, and moreover, $s_p[X \setminus \{B\}] \preceq t_p[X \setminus \{B\}]$ is the most general such pattern. In particular, $r \models (X \setminus \{B\} \rightarrow B, (s_p[X \setminus \{B\} \parallel c_B))$; from this it follows that $(B, c_B) \notin C^+(X, s_p)$.
- If (B, c_B) is removed in 2.c and $B \notin X$, then there exists $C \neq B$, $C \in X$ such that $I \models (R : [X \setminus \{C\}] \rightarrow C, (t_p \parallel t_p[C]))$, where t_p is the most general pattern again. An argument similar to the one given above suffices to show that $(B, c_B) \notin \mathcal{C}^+(X, s_p)$.
- If (B, c_B) is removed in 2.c, then $r \models (X \rightarrow B, (t_p \parallel c_B))$, where t_p is the most general such pattern $(s_p \preceq t_p)$ again. By definition, $(B, c_B) \notin C^+(X, s_p)$.

As a result, the algorithm correctly updates $\mathcal{C}^+(X, t_p)$. $\square$

Implementation details. We now briefly elaborate on the implementation of CTANE. There are four primary computational aspects important for an efficient implementation: (i) the maintenance of the sets $C^+(X, s_p)$ (Step 1); (ii) the validation of the candidate minimal CFDs (Step 2.b); (iii) the generation of $L_{\ell+1}$ (Step 4); and (iv) the checking of support when discovering k-frequent CFDs (Step 4.b(ii)).

The technique underlying (i) and (ii) is based on so-called partitions. More specifically, given (X, s_p) we say that two tuples $u, v \in r$ are equivalent *w.r.t.* to (X, s_p) if $u[X] = v[X] \leq s_p[X]$. Any (X, s_p) therefore induces an equivalence relation on a subset of r. If we denote by $[u]_{(X,s_p)}$ the set of tuples in r that are equivalent with u, then we can use $\pi_{(X,s_p)} = \{[u]_{(X,s_p)} \mid u \in r\}$ to partition a subset of r by (X, s_p). The validity of a CFD $\phi = (X \rightarrow A, (s_p \parallel c_A))$ in r can now be tested by checking whether $|\pi_{(X,s_p)}| = |\pi_{([X,A],(s_p,c_A))}|$. That is, the number of equivalence classes remains the same. It is this characterization of the validity of a CFD that provides an efficient implementation of (ii). Moreover, $\pi_{(X,s_p)}$ can be used to eliminate redundant elements in $C^+(X, s_p)$, making this list as small as possible. In contrast, a naive implementation of Step 1 might keep around elements that never appear together with (X, s_p) in r.

Regarding (iii), we adopt a similar techniques used in TANE to generate partitions corresponding to elements in $L_{\ell+1}$ as the product of previously computed partitions. Moreover, to generate $L_{\ell+1}$, we store elements in L_ℓ lexicographically so that one can efficiently generate candidate patterns (Z, u_p).

Finally, for k-frequent CFDs, partitions can be used efficiently to check the support of a newly created element (Z, u_p) in Step 4.b(ii). Moreover, when (Z, u_p) is obtained from $X \cup Y$ and $u_p = (s_p, t_p[Y_n])$ with $t_p[Y_n] = \_$, we can avoid checking $\mathsf{supp}(Z^c, u_p^c, r)$ altogether. Indeed, the support of this pattern is equal to the support of $\mathsf{supp}(X, s_p, r)$, which is assumed to be k-frequent already since it must belong to L_ℓ (Step 4.b(iii)).

5 FastCFD: A DEPTH FIRST APPROACH

In this section we present FastCFD, an alternative algorithm for discovering minimal, k-frequent (variable *and* constant) CFDs. Given an instance r and a support threshold k, FastCFD finds a canonical cover of all minimal CFDs φ such that $\sup(\varphi, r) \geqslant k$. In contrast to the breadth-first approach of CTANE, FastCFD discovers k-frequent minimal CFDs in a *depth-first* way. It is inspired by FastFD [14], a depth-first algorithm for discovering FDs.

FastCFD first decomposes the problem of finding a canonical cover by finding canonical covers consisting of CFDs with a specified RHSattribute. More specifically, for each attribute A in $\mathbf{attr}(R)$, FastCFD looks for all CFDs of the form $\varphi = (Y \to A, t_p)$ such that $Y \subseteq \mathbf{attr}(R) \setminus \{A\}$, φ is minimal, and moreover $\sup(\varphi, r) \geqslant k$. We denote this set of CFDs by $\mathsf{Cover}(A, r, k)$. Clearly, all k-frequent minimal CFDs in r can then be obtained as $\bigcup_{A \in \mathbf{attr}(R)} \mathsf{Cover}(A, r, k)$. The technical challenge of FastCFD therefore shifts to the computation of $\mathsf{Cover}(A, r, k)$ for a given $A \in \mathbf{attr}(R)$, r and $k \geqslant 0$.

It is to compute $\mathsf{Cover}(A, r, k)$ that FastCFD leverages a depth-first search strategy. More specifically, the key observation behind FastCFD is a relationship between CFDs $\varphi = (Y \to A, t_p)$ in $\mathsf{Cover}(A, r, k)$ and so-called *covers of difference sets*. Intuitively, by using the difference sets of r with respect to an attribute A and pattern tuple t_p, we identify those attributes (including the attribute A) in which pairs of tuples in r that match the pattern tuple may possible differ. A cover of these difference sets contains at least one attribute for each pair of tuples. As we will show below (Lemma 4), the minimal covers of the difference sets correspond to the left-hand sides of (minimal) CFDs in $\mathsf{Cover}(A, r, k)$. Therefore, FastCFD needs to find all minimal covers of the difference sets with respect to A and all pattern tuples t_p.

These minimal covers are computed by a procedure, referred to as FindCover. In a nutshell, procedure FindCover loops over all relevant pattern tuples (X, t_p) (as we will see below it suffices to consider free itemsets only). For each (X, t_p), it invokes a recursive procedure, denoted by FindMin. This procedure will extend X by all subsets Y in $\mathbf{attr}(R) \setminus \{X \cup A\}$, and test whether the resulting CFD $([X, Y] \to (t_p, \_, \ldots, \_ | t_a))$ is minimal. To do this it leverages the relationship with difference sets to optimally prune subsets that do not lead to minimal CFDs. As will be explained in more detail below, FindMin uses a depth-first, left-to-right traversal of the space of subsets of $\mathbf{attr}(R) \setminus \{X \cup A\}$.

Before we present FastCFD, we first define difference sets and develop a pattern pruning strategy.

Difference sets. As previously mentioned, to compute $\mathsf{Cover}(A, r, k)$ in a depth-first way, we need the notion of difference sets. Similar to [14], we define the *difference set* for a pair of tuples $t_1, t_2 \in r$ by

$$\mathfrak{D}(t_1, t_2; r) = \{B \in \mathbf{attr}(R) \mid t_1[B] \neq t_2[B]\},$$

i.e., the set of attributes in which t_1 and t_2 differ. We define the difference set of r to be $\mathfrak{D}(r) = \{\mathfrak{D}(t_1, t_2; r) \mid t_1, t_2 \in r\}$.

We denote by $\mathfrak{D}_A(r)$ the set $\{Y \setminus \{A\} \mid Y \in \mathfrak{D}(r), A \in Y\}$, *i.e.,* the set of attribute sets $Y \setminus \{A\}$ such that there exist tuples in r that disagree on all of the attributes in Y, including A.

A difference set $Y \in \mathfrak{D}_A(r)$ is said to be *minimal* if for all $Y' \in \mathfrak{D}_A(r)$ such that if $Y' \subseteq Y$ then $Y' = Y$. We denote the set of minimal difference sets in $\mathfrak{D}_A(r)$ by $\mathfrak{D}_A^m(r)$.

To characterize the relationship between minimal difference sets of minimal CFDs in $\mathsf{Cover}(A, r, k)$, we need the following notations. Denote by $\mathcal{P}(\mathbf{attr}(R))$ the power set of $\mathbf{attr}(R)$. Let $Z \subseteq \mathbf{attr}(R)$ and $X \subseteq \mathcal{P}(\mathbf{attr}(R))$. We say that Z *covers* X iff for each $Y \in X$, $Y \cap Z \neq \emptyset$. Moreover, Z is a *minimal cover* for X if no $Z' \subset Z$ covers X.

The relationship between difference sets and the validity of CFDs is given in the lemma below. Recall that for a pattern t_p, we denote by r_{t_p} the set of tuples in r that match with t_p.

Lemma 4:
(a) *For any constant* CFD $\phi = (X \to A, (t_p \parallel a))$, $r \models \phi$ *and* $\sup(\phi, r) \geq k$ *iff* $|r_{t_p}| \geq k$, $\mathfrak{D}_A^m(r_{t_p}) = \emptyset$, *and* $\pi_A(r_{t_p}) = (a)$.
(b) *For any variable* CFD $\phi = (X \to A, (t_p \parallel \_))$, $r \models \phi$ *and* $\sup(\phi, r) \geq k$ *iff* $|r_{t_p}| \geq k$ *and* X *covers* $\mathfrak{D}_A^m(r_{t_p})$. □

Proof: This follows immediately from the semantics of CFDs and the definition of minimal covers of difference sets. □

Lemma 4 provide a means of testing whether a CFD holds in terms of difference sets. Furthermore, it also forms the basis for finding *minimal* k-frequent CFDs. Indeed, consider constant CFDs. To find a minimal k-frequent constant CFD $(X \to A, (t_p \parallel a))$, Lemma 4 tells us that we need to find a k-frequent itemset (X, t_p) in r, such that $\mathfrak{D}_A^m(r_{t_p}) = \emptyset$ and $\mathfrak{D}_A^m(r_{t_p}[X']) \neq \emptyset$ for any $X' \subset X$ of size $|X| - 1$. The constant a is then given by $\pi_A(r_{t_p})$. We refer to this condition on (X, t_p) as condition (a).

Next, consider variable CFDs $(X \to A, (t_p \parallel \_))$. Observe that sets in $\mathfrak{D}_A^m(r_{t_p})$ only contain attributes B for which $t_p[B] = \_$. It is therefore sufficient to only consider constant pattern tuples in the difference sets. We denote by $X^c \subseteq X$ the set of attributes in X such that $t_p[X^c]$ consists of constants only. The corresponding pattern tuple $t_p[X^c]$ is denoted by t_p^c. We use X^v to denote the remaining attributes in $X \setminus X^c$, and $t_p^v = (\_, \ldots, \_)$ to denote pattern tuple $t_p[X \setminus X^c]$.

Hence, to find a minimal k-frequent variable CFD $([X^c, X^v] \to A, (t_p^c, t_p^v \parallel \_))$ we have to find a k-frequent itemset (X^c, t_p^c) in r such that
(b1) X^v is a minimal cover of $\mathfrak{D}_A^m(r_{t_p^c})$, *i.e.,* there exists no $Y' \subseteq X^v$ of size $|X^v| - 1$ that covers $\mathfrak{D}_A^m(r_{t_p^c})$; and
(b2) none of the constants in t_p^c can be replaced by a '_', *i.e.,* there exists no $X' \subseteq X^c$ of size $|X^c| - 1$ such that $X^v \cup (X^c \setminus X')$ covers $\mathfrak{D}_A^m(r_{t_p^c[X']})$.

Conditions (b1) and (b2) are on X^v and X^c, respectively. They guarantee that $([X^c, X^v] \rightarrow A, (t_p^c, t_p^v \parallel \_))$ is left-reduced.

Procedure FindCover uses a depth-first exploration of all subsets of $\mathbf{attr}(R) \setminus \{A\}$ to find minimal covers of the difference sets $\mathfrak{D}_A^m(r_{t_p})$ for pattern tuples t_p satisfying the conditions (a), (b1) and (b2) described above. Before we present FindCover in more detail, we describe an additional optimization when discovering variable CFDs.

Efficient pattern pruning strategy. We have seen that a minimal k-frequent variable CFDs is of the form $([X^c, X^v] \rightarrow A, (t_p^c, t_p^v \parallel \_))$, where (X^c, t_p^c) is a k-frequent itemset. Similar to the constant CFD case (see Proposition 1) we now show that it is not necessary to consider *all* k-frequent itemsets (X^c, t_p^c) when discovering minimal variable CFDs. Indeed, the following lemma tells us that it suffices to consider only k-frequent *free* itemsets. This yields a pruning strategy, *i.e.,* by only considering free itemsets. As we will seen in Section 6, the strategy substantially reduces the number of constant pattern candidates and significantly improves the efficiency of CFD discovery.

Lemma 5: *Let* $\phi = (X \rightarrow A, (t_p \parallel \_))$ *be a variable* CFD *such that* $r \models \phi$ *and* $\sup(\phi, r) \geq k$. *If* ϕ *is* minimal *then the constant pattern in* t_p*, denoted by* (X^c, t_p^c)*, is a* k*-frequent free itemset.* □

Proof: Since $\sup(\phi, r) \geq k$, (X^c, t_p^c) is k-frequent. It suffices to show that (X^c, t_p^c) is a free itemset.

Suppose by contradiction that (X^c, t_p^c) is not a free itemset. Then there must exist a free itemset (Y, s_p) such that $\sup(X^c, t_p^c, r) = \sup(Y, s_p, r) \geq k$, and $(X^c, t_p^c) \prec (Y, s_p)$. If $(X^c, t_p^c) \prec (Y, s_p)$, then by Proposition 1, there must exist constant CFDs $\phi^{'} = (Y \rightarrow B, (s_p \parallel b))$ for any $B \in X^c \setminus Y$ and $b \in t_p^c \setminus s_p$. We know that $\phi = (X \rightarrow A, (t_p \parallel \_))$ holds on r. Let X^v denote the set of attributes $X \setminus X^c$, and t_p^v the pattern $t_p \setminus t_p^c$. Then because ϕ and $\phi^{'}$ hold on r, we have that $r \models (Y \cup X^v \rightarrow A, (s_p \cup t_p^v \parallel \_))$. This contradicts the condition that ϕ is *minimal* because $Y \cup X^v \subset X$, *i.e.,* (X^c, t_p^c) is not the "most general" constant pattern for ϕ. Thus (X^c, t_p^c) must be a k-frequent free itemset. □

Algorithm FastCFD. We next describe algorithm FastCFD and its component procedures FindCover and FindMin in more detail. As previously mentioned, given r and $k \geqslant 0$, FastCFD calls FindCover(A, r, k) for each attribute $A \in \mathbf{attr}(R)$. The final result is the union of Cover(A, r, k), for each $A \in \mathbf{attr}(R)$, as returned by FindCover.

Algorithm FindCover. Procedure FindCover(A, r, k) in turn invokes the procedure FindMin. More specifically, Proposition 1 and Lemma 5 state that it suffices to consider k-frequent free itemsets as constant patterns of CFDs only. Hence, FindCover first extracts the set of the k-frequent *free* itemsets $\mathsf{Fr}_k(r)$ of r, in which itemsets are kept in the ascending order *w.r.t.* their sizes. To efficiently retrieve elements in $\mathsf{Fr}_k(r)$, FindCover indexes those itemsets in a hash table. Second, for each (X, t_p) in $\mathsf{Fr}_k(r)$, FindCover maintains $\mathfrak{D}_A^m(r_{t_p})$, *i.e.,* the set of *minimal* difference sets produced from all tuples in r_{t_p}. Then for a given $(X, t_p) \in \mathsf{Fr}_k(r)$, FindCover recursively calls FindMin to find a minimal cover Y of $\mathfrak{D}_A^m(r_{t_p})$ and tests conditions (a), (b1) and (b2) described above.

Algorithm FindMin. Procedure FindMin finds the minimal covers by traversing all subsets of $\mathbf{attr}(R) \setminus \{A\}$ in a depth-first way. That is, we assume an ordering $<_{\mathbf{attr}}$ on $\mathbf{attr}(R)$. All subsets of $\mathbf{attr}(R) \setminus \{A\}$ are then enumerated in a *depth-first*, *left-to-right* fashion based on the given attribute ordering. For instance, suppose that $\mathbf{attr}(R) = \{A, B, C, D\}$ and $A <_{\mathbf{attr}} B <_{\mathbf{attr}} C <_{\mathbf{attr}} D$. Then, starting from the empty set, the subsets of $\mathbf{attr}(R) \setminus \{A\}$ are generated in the following order: $\{B\}$, $\{B, C\}$, $\{B, C, D\}$, $\{B, D\}$, $\{C\}$, $\{C, D\}$ and $\{D\}$. It is common to represent these sets in an enumeration tree according to $<_{\mathbf{attr}}$, in which each set corresponds to a path from the root, ending in the node representing that set. For instance, $\{B, C\}$ corresponds to a path $\emptyset, B, C$ in the enumeration tree. In the following we abuse notation and represent both the set $Y \subseteq \mathbf{attr}(R)$ and its corresponding path in the tree by Y.

During the enumeration of the subsets by FindMin, we denote by $Y \subseteq \mathbf{attr}(R)$ the current path in the enumeration tree. Furthermore, when inspecting Y, FindMin maintains the difference sets in $\mathfrak{D}_A^m(r_{t_p})$ that are *currently not covered yet* by attributes in Y. We denote this set by $\mathfrak{D}_A^m(r_{t_p})[Y]$. Initially, *i.e.,* when $Y = \emptyset$, this set is equal to $\mathfrak{D}_A^m(r_{t_p})$. The details of FindMin are as follows:

Input: $A \in \mathbf{attr}(R)$, $(X, t_p) \in \mathsf{Fr}_k(r)$, $Y \subseteq \mathbf{attr}(R) \setminus \{A\}$, $\mathfrak{D}_A^m(r_{t_p})[Y]$, and $<_{\mathbf{attr}}$.

Output: minimal CFDs $\varphi = ([X, Y] \rightarrow A, (t_p, \_, \ldots \_ \parallel t_a))$, where t_a is a constant or "_".

Base case:

1) If $\emptyset \in \mathfrak{D}_A^m(r_{t_p})[Y]$, then return an empty set. By Lemma 4, $([X, Y], (t_p, \_, \ldots, \_))$ can never lead to a valid CFD.
2) If Y contains the last attributes in $\mathbf{attr}(R) \setminus \{A\}$ *w.r.t.* $<_{\mathbf{attr}}$, but $\mathfrak{D}_A^m(r_{t_p})[Y] \neq \emptyset$, then return an empty set. By Lemma 4, $r \not\models ([X, Y] \rightarrow A, (t_p, \_, \ldots, \_ \parallel \_))$ because Y does not cover $\mathfrak{D}_A^m(r_{t_p})$; moreover, since $([X, Y], (t_p, \_, \ldots, \_))$ cannot be further extended, this pattern does not lead to a valid CFD.
3) If $\mathfrak{D}_A^m(r_{t_p})[Y] = \emptyset$, then Y is a cover of $\mathfrak{D}_A^m(r_{t_p})$. There are two cases to consider corresponding to the conditions (a) and (b1–b2).
   a) If $\mathfrak{D}_A^m(r_{t_p}) = \emptyset$, then by Lemma 4, there exists a constant t_a, $r \models (X \rightarrow A, (t_p \parallel t_a))$. In order to check for minimality, we need to verify whether there is no $X^{'} \subset X$ of size $|X| - 1$ such that $r \models (X^{'} \rightarrow A, (t_p[X^{'}] \parallel t_a))$. If this holds, then output *constant* CFD $(X \rightarrow A, (t_p \parallel t_a))$.
   b) if $\mathfrak{D}_A^m(r_{t_p}) \neq \emptyset$, then Lemma 4 implies that $r \models ([X, Y] \rightarrow A, (t_p, \_, \ldots, \_ \parallel \_))$. In order to check for minimality, we need to verify whether
      i) there is no $Y^{'} \subset Y$ of size $|Y| - 1$ such that $Y^{'}$ covers $\mathfrak{D}_A^m(r_{t_p[X]})$;
      ii) there is no $X^{'} \subset X$ of size $|X| - 1$ such that $Y \cup (X \setminus X^{'})$ covers $\mathfrak{D}_A^m(r_{t_p[X^{'}]})$.

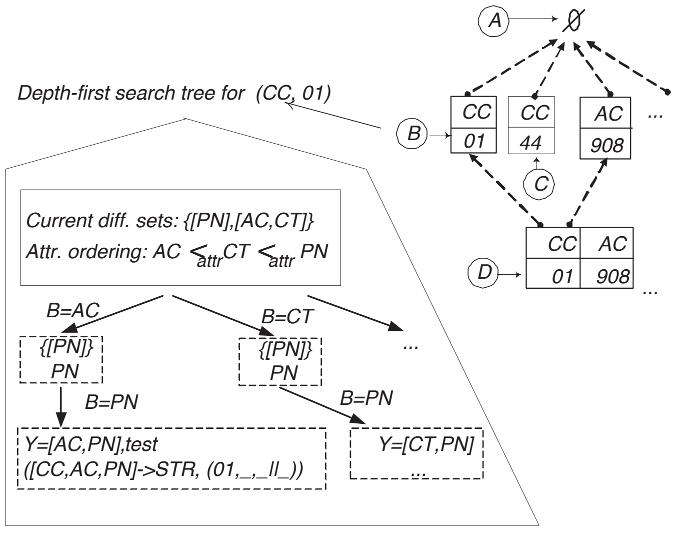
A
∅
B
C
D
CC 01
CC 44
AC 908
CC AC 01 908
...
Depth-first search tree for (CC, 01)
Current diff. sets: {[PN],[AC,CT]}
Attr. ordering: AC <attr CT <attr PN
B=AC
B=CT
{[PN]} PN
{[PN]} PN
B=PN
B=PN
Y=[AC,PN],test ([CC,AC,PN]->STR, (01,_,_ll_))
Y=[CT,PN]

Fig. 4. Partial execution of FindCover. Explanation of circled A, B, C and D are provided in Example 9.

If conditions (i) and (ii) are both satisfied, then output *variable* CFD $([X, Y] \rightarrow A, (t_p, \_, \ldots, \_ \parallel \_))$.

Recursive case:

4) For each attribute B that appears after Y *w.r.t.* $<_{\mathbf{attr}}$, we do the following.
   a) Let $Y^{'} = Y \cup \{B\}$ and $\mathfrak{D}^m_A(r_{t_p})[Y']$ be the difference sets of $\mathfrak{D}^m_A(r_{t_p})[Y]$ not covered by B.
   b) Call FindMin$(A, (X, t_p), Y^{'}, \mathfrak{D}^m_A(r_{t_p})[Y'], <_{\mathbf{attr}})$ recursively following the depth-first strategy.

Before illustrating algorithm FastCFD, we remark the following. A careful reader might wonder how $\mathfrak{D}^m_A(r_{t_p[X']})$ is obtained in Step 3.b(ii). After all, the only difference sets that are maintained are those related to k-frequent free patterns. Note, however, that $(X^{'}, t_p[X^{'}])$ is a k-frequent itemset due to the *anti-monotonicity* property of frequent itemsets. Furthermore, there exist k-frequent free itemsets (Z, s_p) such that $(Z, s_p) \preceq (X^{'}, t_p[X^{'}])$. Because $|\mathsf{supp}(X^{'}, t_p[X^{'}])| = \max\{|\mathsf{supp}(Z, s_p)|\}$, $\mathfrak{D}^m_A(r_{t_p[X']})$ is the same as $\mathfrak{D}^m_A(r_{s_p[Z]})$, where (Z, s_p) is the free itemset with the maximum cardinality for all $(Z, s_p) \preceq (X^{'}, t_p[X^{'}])$. Since $\mathfrak{D}^m_A(r_{s_p[Z]})$ *is* maintained, we use this set in Step 3.b(ii).

Example 9: Consider again the cust relation of Fig. 1. We give a partial run of FindCover$(\mathbf{attr}(R) \setminus \mathsf{STR}, \mathsf{STR}, \mathsf{cust}, 2)$ involving only attributes CC, AC, PN, CT, ZIP and STR. We leave out attribute NM to simplify the discussion. We assume a support threshold $k = 2$ and assume that $<_{\mathbf{attr}}$ orders attributes alphabetically. Figure 4 depicts the following: in the top right corner we show a partial set of the k-frequent free itemsets in $\mathsf{Fr}_k(r)$, where the immediate subsets of the free itemsets are illustrated by dotted arrows; in the bottom left corner we draw a partial execution tree of FindMin for the free pattern $(\mathsf{CC}, 01)$. We highlight circled points A, B, C and D. Note that the execution of FindCover constructs a depth-first search tree for every free pattern although only the one for $(\mathsf{CC}, 01)$ is shown in the figure.

(A) As outlined above, FindCover passes free patterns to FindMin. Assume that the first free pattern to be considered is $\emptyset$. In this case, the execution of FindMin is exactly the same as in the FastFD algorithm [14]. After this step all minimal FDs of the form $(Y \rightarrow \mathsf{STR}, (\_, \ldots, \_ \parallel \_))$ are returned.

(B) Next, we consider the free pattern $(\mathsf{CC}, 01)$ with $r_{\mathsf{CC}=01} = \{t_1, t_2, t_3, t_4, t_8\}$, and *minimal* difference sets

$$\mathfrak{D}^m_{\mathsf{STR}}(r_{\mathsf{CC}=01}) = \{[\mathsf{PN}], [\mathsf{AC}, \mathsf{CT}]\}.$$

Hence, FindMin will find a cover for $\mathfrak{D}^m_{\mathsf{STR}}(r_{\mathsf{CC}=01})$ by the recursive process invoked in Step 4. The (partial) enumeration tree of the subsets of $\{\mathsf{AC}, \mathsf{CT}, \mathsf{PN}\}$ according to $<_{\mathbf{attr}}$ is illustrated in Fig. 4. The corresponding covers Y of $\mathfrak{D}^m_{\mathsf{STR}}(r_{\mathsf{CC}=01})$ computed are are $[\mathsf{AC}, \mathsf{PN}]$ and $[\mathsf{CT}, \mathsf{PN}]$. Consider the cover $[\mathsf{AC}, \mathsf{PN}]$ and its minimal CFD candidate

$$\phi^{'} = ([\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \rightarrow \mathsf{STR}, (01, \_, \_ \parallel \_))$$

in Step 3.b. Although the algorithm verifies that $\phi^{'}$ is minimal for $r_{\mathsf{CC}=01}$ in Step 3.b(i), it still needs to inspect whether $[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}]$ covers $\mathfrak{D}^m_{\mathsf{STR}}(r_\emptyset)$ in Step 3.b(ii), where $\emptyset$ is the only immediate subset of pattern $(\mathsf{CC}, 01)$. In this case, it finds out that $[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}]$ covers $\mathfrak{D}^m_{\mathsf{STR}}(r_\emptyset)$, which indicates that $r \models ([\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \rightarrow \mathsf{STR}, (\_, \_, \_ \parallel \_))$. Thus $\phi^{'}$ is not a minimal CFD.

(C) Similarly, consider the free pattern $(\mathsf{CC}, 44)$ with $r_{\mathsf{CC}=44} = \{t_5, t_6, t_7\}$ and minimal difference sets

$$\mathfrak{D}^m_{\mathsf{STR}}(r_{\mathsf{CC}=44}) = \{[\mathsf{AC}, \mathsf{CT}, \mathsf{ZIP}]\}.$$

The covers of $\mathfrak{D}^m_{\mathsf{STR}}(r_{\mathsf{CC}=44})$ are AC, CT, and ZIP. For the cover AC, FindMin needs to inspect if its CFD

$$\phi = ([\mathsf{CC}, \mathsf{AC}] \rightarrow \mathsf{STR}, (44, \_ \parallel \_))$$

is minimal. In Step 3.b(i), it verifies that ϕ is minimal for $r_{\mathsf{CC}=44}$, but it still needs to inspect whether $[\mathsf{CC}, \mathsf{AC}]$ covers $\mathfrak{D}^m_{\mathsf{STR}}(r_\emptyset)$ (*i.e.*, $\mathfrak{D}^m_{\mathsf{STR}}(r)$) in Step 3.b(ii), where again $\emptyset$ is the only immediate subset of pattern $(\mathsf{CC}, 44)$. As we can observe from the cust relation, $\mathfrak{D}(t_2, t_4) = \{\mathsf{PN}, \mathsf{STR}\}$, and $[\mathsf{PN}] \in \mathfrak{D}^m_{\mathsf{STR}}(r)$ (one may compute $\mathfrak{D}^m_{\mathsf{STR}}(r)$ to verify that $[\mathsf{PN}] \in \mathfrak{D}^m_{\mathsf{STR}}(r)$). This implies that $[\mathsf{CC}, \mathsf{AC}]$ cannot be a cover for $\mathfrak{D}^m_{\mathsf{STR}}(r)$. Thus, ϕ is a minimal CFD.

(D) As a final example we consider the free set $(X, t_p) = ([\mathsf{CC}, \mathsf{AC}], [01, 908])$ with $r_{t_p} = \{t_1, t_2, t_4\}$ and *minimal* difference sets:

$$\mathfrak{D}^m_{\mathsf{STR}}(r_{t_p}) = \{[\mathsf{PN}]\}.$$

The corresponding cover of $\mathfrak{D}^m_{\mathsf{STR}}(r_{t_p})$ is $[\mathsf{PN}]$. Consider its minimal CFD candidate

$$\phi^{''} = ([\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \rightarrow \mathsf{STR}, (01, 908, \_ \parallel \_))$$

in Step 3.b. Although FindMin verifies that $\phi^{''}$ is minimal for $r_{t_p^c}$ in Step 3.b(i), it still needs to inspect all immediate subsets of $([\mathsf{CC}, \mathsf{AC}], [01, 908])$, *i.e.*, $(\mathsf{CC}, 01)$ and $(\mathsf{AC}, 908)$, for the minimality of $\phi^{''}$. Suppose that FindMin inspects $(\mathsf{CC}, 01)$ first. It finds out that $[\mathsf{AC}, \mathsf{PN}]$ is actually a cover for $\mathfrak{D}^m_{\mathsf{STR}}(r_{\mathsf{CC}=01})$. Thus $\phi^{''}$ is not a minimal CFD. □

Implementation details and optimizations. The key differences between FastCFD and its FD-counterpart FastFD consists of the following: (1) the more complicated condition for testing the validity of a minimal CFD ϕ in terms of the minimality of the *constant pattern* and *unnamed variables* in LHS(ϕ); and (2) the fact that we discover k-frequent CFDs instead of 1-frequent FDs only. Whereas for FDs, the only difference sets needed are $\mathfrak{D}^m_A(r)$ for $A \in \mathbf{attr}(R)$, Lemma 4 states that for CFDs, difference sets $\mathfrak{D}^m_A(r_{t_p})$ are needed for all r_{t_p}, where t_p is a k-frequent free pattern in r. Worse still, when (X, t_p) is reached, the depth-first approach enforces FindMin to use $\mathfrak{D}^m_A(r_{t_p[X']})$ during the minimality check for all $X' \subseteq X$ of size $|X| - 1$. These suggest that we need a very efficient way to compute difference sets. To do so the following two approaches are implemented and evaluated.

NaiveFast. The first one is inspired by the stripped partition idea used by FastFD [14]. Here for a given (X, t_p), the stripped partition of r_{t_p} *w.r.t.* an attribute A is the partition of r_{t_p} *w.r.t.* A from which all single-tuple equivalence classes are removed (see Section 4 for the definition of partition). The computation of the stripped partitions of r_{t_p} for each $A \in \mathbf{attr}(R)$ provides sufficient information to infer for any two tuples on which attributes they agree. By taking complements, one can then infer the difference sets. It should be mentioned that the stripped partitions are often much smaller than the instances, making this approach efficient. We refer to the version that relies on the partition-based approach as NaiveFast.

FastCFD. The second approach relies on the availability of Closed$_2(r)$, which consists of all 2-frequent closed itemsets in r. Given (X, t_p), we can again infer for any two tuples in r_{t_p} on which attributes they agree. Indeed, these sets of attributes are given by the attributes in those itemsets in Closed$_2(r)$ that match t^c_p (the constant part of t_p). By taking the complement we can infer the desired difference sets efficiently. Our experimental study (Section 6) shows that this approach outperforms the partition-based approach and is therefore used to implement difference sets in FastCFD.

Finally, since CFDMiner produces Closed$_k(r)$ as a side-product, we use CFDMiner for constant CFD discovery and FastCFD for variable CFDs only. To do so, we eliminate Step 3.a in FindCover. Taken together, these lead to significant improvements in efficiency, as will be seen in the next section.

Dynamic attribute reordering. Similar to FastFD, FastCFD is equipped with a dynamic reordering of the attributes when enumerating the subsets in the procedure FindMin. More specifically, instead of keeping $<_{\mathbf{attr}}$ fixed throughout the execution of FindMin, an additional step (between Step 4.a and Step 4.b) is included in which the remaining attributes are reordered based on a cost model.

FastCFD employs a cost model similar to FastFD, to dynamically reorder attributes such that attributes that cover the most difference sets are treated first. We refer to [14] for more details concerning the cost model.

6 Experimental Study

We next present an experimental study of our algorithms for discovering minimal CFDs: CFDMiner, CTANE, NaiveFast, FastCFD given in Sections 3, 4 and 5, respectively. We investigate the effects of the following factors on the scalability and the number of minimal CFDs produced: (a) the support threshold k, (b) the size DBSIZE of a sample relation r, *i.e.,* the number of tuples in r, (c) the arity ARITY of r, *i.e.,* the number of columns in r, (d) a *correlation factor* (CF) [14], which indicates that the average range of distinct values in an attribute domain is CF $*$ DBSIZE.

6.1 Experimental Settings

The experiments were conducted on both real-life data and on synthetic datasets generated using real data. Our experiments used real datasets from the UCI machine learning repository (http://archive.ics.uci.edu/ml/), namely, the Wisconsin breast cancer (WBC) and Chess datasets. The following table describes the parameters of those datasets:

Dataset	Arity	Size (# of tuples)
Wisconsin breast cancer (WBC)	11	699
Chess	7	28,056
Tax	14	20,000

To evaluate the scalability, we also used an extension of the relation in Fig. 1, which is a synthetic dataset for tax records generated by populating the database with data used in [1], via a generator. The generator takes parameters ARITY, DBSIZE and CF, and produces datasets accordingly.

The algorithms have been implemented in C++. The program has been tested on *AMD Opteron* Processor (2.6GHZ) with 32GB of memory running the Linux operating system. Our algorithms run entirely in main memory. Each experiment was repeated over 5 times and the average is reported here.

6.2 Experimental Results

We first present our experimental results on generated data, and then our results with real-life data.

<u>**Scalability Experiments**</u>. We study the performance of our algorithms by varying DBSIZE, ARITY, CF, and support threshold k in this set of experiments.

Scalability *w.r.t.* DBSIZE: Fixing ARITY $= 7$ and CF $= 0.7$ we varied DBSIZE from $20K$ to 1 million tuples. We kept *support ratio* SUP%, which is defined as $\frac{k}{\text{DBSIZE}}$, at 0.1%. The response times of our algorithms are reported in Fig. 5. In particular, CFDMiner(2) indicates CFDMiner with $k = 2$, which is used in FastCFD for optimization.

The results of Fig. 5 tell us the following. (a) CFDMiner, which only mines constant CFDs, is multiple orders of magnitude faster than the other algorithms that discover both constant and variable CFDs. (b) The naive version of FastCFD, NaiveFast, outperforms CTANE when DBSIZE is small. However, it does not scale well *w.r.t.* DBSIZE. For example, it outperforms CTANE when DBSIZE is less than $100K$. But when DBSIZE is $300K$, it is 2.5 times slower than CTANE. This behavior is primarily due to the cost incurred in the construction of the difference sets in NaiveFast. As observed for FastFD [14], the difference set construction contributes most to the cost of NaiveFast. When DBSIZE becomes larger, there are more itemsets with large support that need to be considered for constructing the difference sets. This results in a

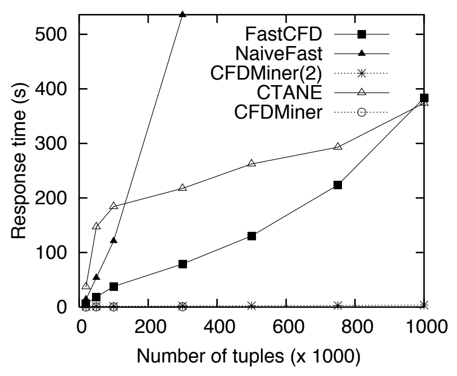

Fig. 5. Scalability *w.r.t.* DBSIZE

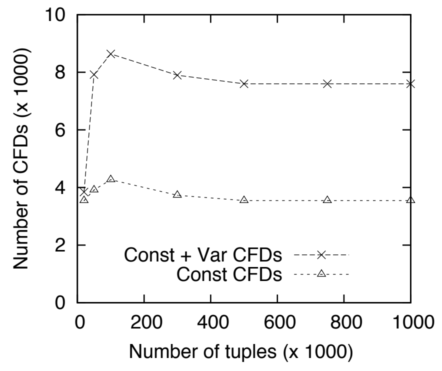

Fig. 6. No. of CFDs found *w.r.t.* DBSIZE

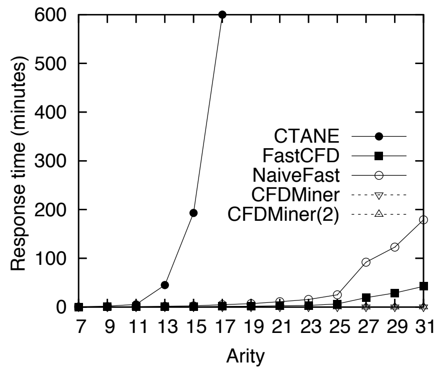

Fig. 7. Scalability *w.r.t.* ARITY

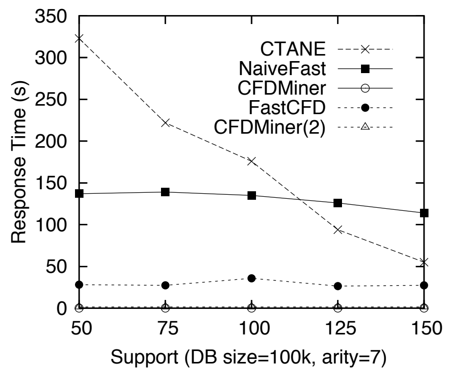

Fig. 8. Scalability *w.r.t.* k

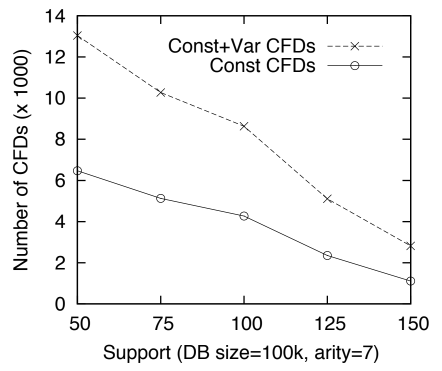

Fig. 9. No. of CFDs found *w.r.t.* k

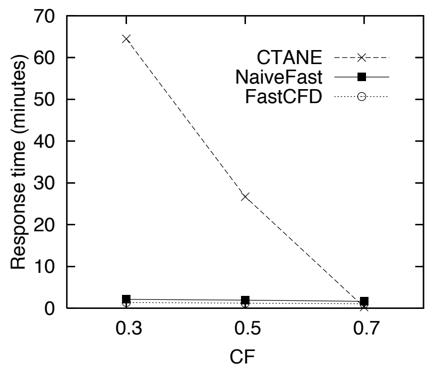

Fig. 10. Scalability *w.r.t.* CF

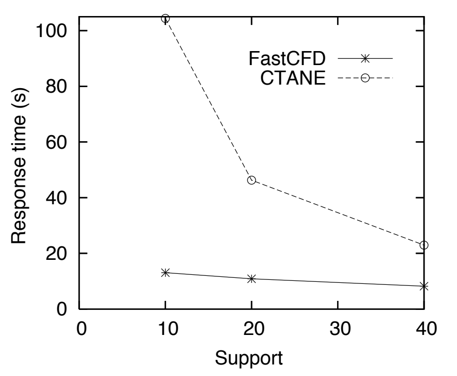

Fig. 11. Wisc. breast cancer

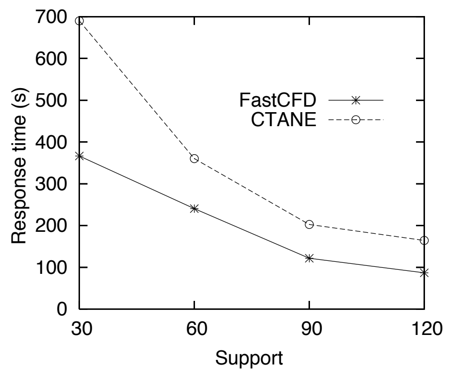

Fig. 12. Chess

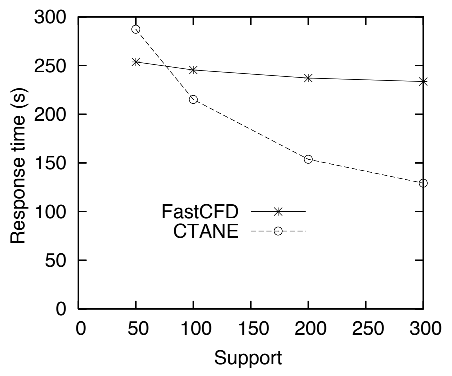

Fig. 13. Tax

significant performance degradation of NaiveFast. (c) FastCFD outperforms CTANE and NaiveFast when DBSIZE is less than one million tuples, which is reasonably large. This verifies the effectiveness of our optimization by leveraging the closed-itemsets from CFDMiner for constructing difference sets.

Figure 6 shows the total number of minimal CFDs discovered by our algorithms. For clarity, only constant and variable CFDs of FastCFD are shown because CTANE, NaiveFast and FastCFD find about the same number of CFDs.

Scalability *w.r.t.* ARITY: Fixing $\mathrm{CF} = 0.7$, $\mathrm{DBSIZE} = 20K$, and $\mathrm{SUP\%} = 0.1\%$, we varied ARITY from 7 to 31. As shown in Fig. 7, CTANE does not scale well with the arity, as expected. In contrast, NaiveFast and FastCFD scale well: both are orders of magnitude better than CTANE when $\mathrm{ARITY} \geq 15$. In addition, FastCFD is 4 times better than NaiveFast when $\mathrm{ARITY} = 31$, which further demonstrates the effectiveness of the optimization techniques of FastCFD via CFDMiner.

Scalability *w.r.t.* k: We fixed $\mathrm{CF} = 0.7$, $\mathrm{DBSIZE} = 100K$, $\mathrm{SUP\%} = 0.1\%$, and varied the support threshold k from 50 to 150. As shown in Fig. 8, NaiveFast and FastCFD only improve slightly when k increases. In contrast, CTANE is highly sensitive to k. For example, NaiveFast outperforms CTANE when k is small (*e.g.,* 50), whereas CTANE outperforms NaiveFast when k is large (*e.g.,* 150). The performance of CTANE improves as k increases. This is because fewer itemsets with large support satisfy k when k becomes larger,

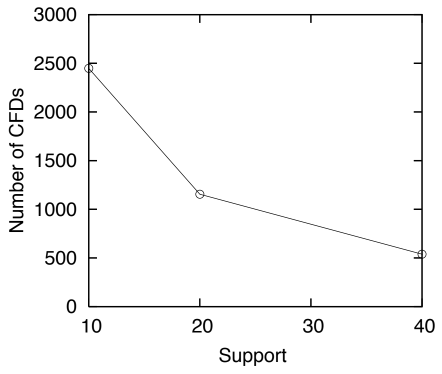

Fig. 14. Wisc. breast cancer

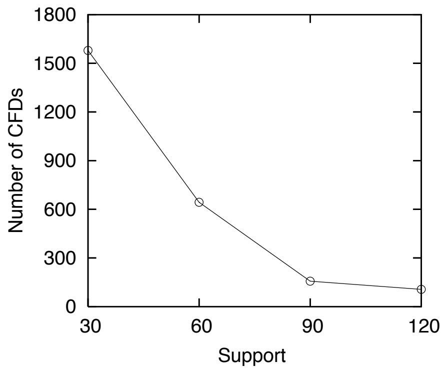

Fig. 15. Chess

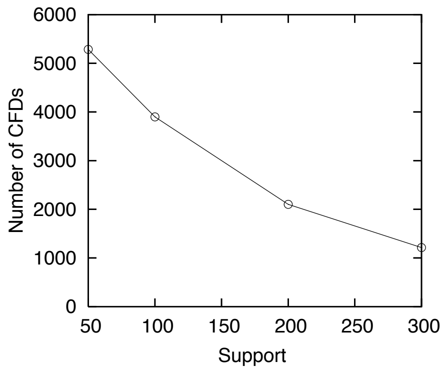

Fig. 16. Tax

which certainly reduces the number of candidates to examine at each level by CTANE. On the other hand, the main cost for NaiveFast and FastCFD is in the construction of difference sets for itemsets with large support, which does not change significantly when k gets larger.

Figure 9 shows that the number of minimal CFDs discovered decreases as k increases, as expected. Again, only constant and variable CFDs of FastCFD are shown because CTANE, NaiveFast and FastCFD find about the same number of CFDs.

Scalability *w.r.t.* CF: We varied CF from 0.3 to 0.7, while fixing $\text{DBSIZE} = 50K$, $k = 50$ and $\text{ARITY} = 9$. As shown in Fig. 10, CTANE is very sensitive to the number of distinct values in an attribute domain. As we fixed the total number of tuples at $50K$, when CF decreases, the number of itemsets with large support increases. For a fixed k, this means more itemsets satisfying the support threshold in CTANE. Thus the algorithm has to examine more candidates at each level, which leads to performance degradation. In contrast, the performance of NaiveFast and FastCFD only degrades slightly as CF decreases.

Real Data Experiments We have conducted experiments on real-life data, including the Chess, WBC, and synthetic Tax datasets. For each dataset, k was varied. Figures 11, 12, 13 show the response times of CTANE and FastCFD when k is varied, while figures 14, 15 and 16 show the corresponding numbers of CFDs discovered by the algorithms. Consistent with our previous experiments, CTANE is sensitive to the support threshold k, and its performance improves when k increases. FastCFD is less sensitive to k, and its performance only improves slightly as k increases. Both algorithms discover fewer number of CFDs as k increases.

Summary. From the experimental results we find the following. (a) Constant CFD discovery (CFDMiner) can be multiple orders of magnitude faster than general CFD discovery (CTANE or FastCFD). (b) CTANE usually works well when the arity of a relation is small and the support threshold is large, but it scales poorly when the arity increases. (c) NaiveFast and FastCFD are far more efficient than CTANE when the arity of the relation is large. (d) Our optimization technique based on closed-itemset mining is effective: FastCFD significantly outperforms NaiveFast, especially when the arity is large.

7 Related Work

Prior work on conditional dependencies has mostly focused on the consistency and implication analyses of CFDs [1], repairing methods to localize and fix errors detected by CFDs [26], propagation of CFDs from source data to views in data integration [27], extensions of CFDs by adding disjunction and negation [28] or adding ranges [10], confidence of CFDs [29], as well as extensions of inclusion dependencies with conditions (referred to as CINDs) [30]. To our knowledge, CFD discovery was only studied in [21], [10], [31]. Except these, the previous work assumes that CFDs are already designed and provided.

As remarked in Section 1, there has been a host of work on minimal FD discovery [12], [13], [14], [15], [16], [17], [18]. Minimal CFDs, however, are more involved than their FD counterparts: they require both the minimality of attributes and the minimality of patterns (Section 2). Our algorithms CTANE and FastCFD extend TANE [13] and FastFD [14], respectively, for discovering minimal CFDs.

Closer to our work are [10], [21], [31]. For a fixed traditional FD, [10] proposed criteria for sensible patterns that, together with the FD, make useful CFDs. It showed that the problem of finding such patterns is NP-complete, and developed efficient heuristic algorithms for discovering patterns from samples. In contrast to [10], this work studies CFD discovery when the embedded traditional FDs are *not* given. An algorithm for discovering CFDs is developed in [21], which aims to find both traditional FDs and patterns in CFDs, the same as what this work does. Several interest measures for discovered CFDs are also proposed there, including support (which we also consider), conviction and χ^2-test. The algorithm of [21] differs from our algorithms in the following aspects. (a) The algorithm of [21] is an extension of TANE [13]; as shown in Section 6, levelwise CFD-discovery algorithms may not scale well with the arity of sample datasets. It is to deal with such data that FastCFD is provided. In contrast, this issue is not addressed in [21]. (b) In addition, constant CFD discovery is not considered in [21], despite its wide applications in data cleaning and data integration. (c) Moreover, [21] does not consider optimizations based on closed itemsets mining, which is employed by FastCFD.

As observed in Sections 1 and 3, CFD discovery is also closely related to (non-redundant) association rule mining [22], [23], [24]. In particular, CFDMiner is based on the mining algorithm proposed in [24]. Recently an algorithm was proposed in [31] for mining association rules of the form $Q_1 \Rightarrow Q_2$, where Q_1, Q_2 are simple conjunctive queries and Q_2 is contained in Q_1. Since CFDs and CINDs can be viewed as such association rules, the algorithm of [31] may be used to mine general CFDs and CINDs. The method of [31], however, can only discover CFDs with 100% confidence, and in addition, the minimality of CFDs is not investigated in [31]. In contrast, the algorithms of this work are developed to discover minimal k-frequent CFDs. The connection between association rule mining and constant CFD discovery is also observed in [32]. Neither [31] nor [32] provides any experimental results.

8 Conclusions

We have developed and implemented three algorithms for discovering minimal CFDs: (1) CFDMiner for mining minimal constant CFDs, a class of CFDs important for both data cleaning and data integration; (2) CTANE for discovering general minimal CFDs based on the levelwise approach; and (3) FastCFD for discovering general minimal CFDs based on a depth-first search strategy, and a novel optimization technique via closed-itemset mining. As suggested by our experimental results, these provide a set of tools for users to choose for different applications. When only constant CFDs are needed, one can simply use CFDMiner without paying the price of mining general CFDs. When the arity of a sample dataset is large, one should opt for FastCFD. When k-frequent CFDs are needed for a large k, one could use CTANE.

There is naturally much to be done. First, we are currently experimenting with various datasets collected from real life. Second, we are investigating how to discover minimal CFDs from a dataset r when *both* its arity *and* its size are large. To our knowledge, no dependency discovery algorithms scale very well in this setting, even those for traditional FDs. One way around this is by sampling r, *i.e.,* to find a subset r_s of r by selectively drawing tuples from r such that r_s accurately represents r and is small enough to be efficiently processed by FastCFD or CTANE. It is, however, nontrivial to find a sampling method with performance guarantee, *i.e.,* to ensure that the estimated inaccuracy rate is below a predefined bound with high confidence. We are experimenting with the stratified sampling method [33] for this purpose. Third, while we have employed in FastCFD techniques for mining closed itemsets, we expect that other mining techniques may also shed light in improving the performance of discovery algorithms. Fourth, we plan to explore the use of CFD inference in discovery, to eliminate CFDs that are entailed by those CFDs already found. Finally, while the focus of this work is on algorithmic issues for mining CFDs, a topic for future work is to assess quality measures for CFDs, including those studied in [10], [21].

References

[1] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *TODS*, vol. 33, no. 2, june 2008.

[2] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: Consistency and accuracy," in *VLDB*, 2007.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," *TPLP*, vol. 3, no. 4-5, pp. 393–424, 2003.

[4] J. Chomicki and J. Marcinkowski, "Minimal-change integrity maintenance using tuple deletions," *Information and Computation*, vol. 197, no. 1-2, pp. 90–121, 2005.

[5] J. Wijsen, "Database repairing using updates," *TODS*, vol. 30, no. 3, pp. 722–768, 2005.

[6] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.

[7] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches." *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, 2000.

[8] Gartner, "Forecast: Data quality tools, worldwide, 2006-2011," 2007.

[9] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[10] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," in *VLDB*, 2008.

[11] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson, "Entity identification in database integration," *Inf. Sci.*, vol. 89, no. 1-2, pp. 1–38, 1996.

[12] H. Mannila and K.-J. Räihä, "Dependency inference," in *VLDB*, 1987.

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

[14] C. M. Wyss, C. Giannella, and E. L. Robertson, "FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract," in *DaWak*, 2001.

[15] P. A. Flach and I. Savnik, "Database dependency discovery: A machine learning approach," *AI Commun.*, vol. 12, no. 3, pp. 139–160, 1999.

[16] S. Lopes, J.-M. Petit, and L. Lakhal, "Efficient discovery of functional dependencies and armstrong relations," in *EDBT*, 2000.

[17] T. Calders, R. T. Ng, and J. Wijsen, "Searching for dependencies at multiple abstraction levels," *TODS*, vol. 27, no. 3, pp. 229–260, 2003.

[18] R. S. King and J. J. Legendre, "Discovery of functional and approximate functional dependencies in relational databases," *JAMDS*, vol. 7, no. 1, pp. 49–59, 2003.

[19] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga, "Cords: Automatic discovery of correlations and soft functional dependencies," in *SIGMOD*, 2004.

[20] H. Mannila and H. Toivonen, "Levelwise search and borders of theories in knowledge discovery," *Data Min. Knowl. Discov.*, vol. 1, no. 3, pp. 259–289, 1997.

[21] F. Chiang and R. Miller, "Discovering data quality rules," in *VLDB*, 2008.

[22] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules," in *Advances in Knowledge Discovery and Data Mining*, 1996.

[23] M. J. Zaki, "Mining non-redundant association rules," *Data Min. Knowl. Discov.*, vol. 9, no. 3, pp. 223–248, 2004.

[24] J. Li, G. Liu, and L. Wong, "Mining statistically important equivalence classes and delta-discriminative emerging patterns," in *KDD*, 2007.

[25] H. Li, J. Li, L. Wong, M. Feng, and Y.-P. Tan, "Relative risk and odds ratio: a data mining perspective," in *PODS*, 2005.

[26] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: Consistency and accuracy," in *VLDB*, 2007.

[27] W. Fan, S. Ma, Y. Hu, J. Liu, and Y. Wu, "Propagating functional dependencies with conditions," in *VLDB*, 2008.

[28] L. Bravo, W. Fan, F. Geerts, and S. Ma, "Increasing the expressivity of conditional functional dependencies without extra complexity," in *ICDE*, 2008.

[29] G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, and X. Zhang, "Estimating the confidence of conditional functional dependencies," in *SIGMOD*, 2009.

[30] L. Bravo, W. Fan, and S. Ma, "Extending dependencies with conditions," in *VLDB*, 2007.

[31] B. Goethals, W. L. Page, and H. Mannila, "Mining association rules of simple conjunctive queries," in *SDM*, 2008.

[32] R. Medina and N. Lhouari, "A unified hierarchy for functional dependencies, conditional functional dependencies and association rules," in *ICFCA*, 2009.

[33] N. Alon and J. H. Spencer, *"The Probabilistic Method"*. John Wiley Inc., 1992.