Discovering Conditional Functional Dependencies

Wenfei Fan, Floris Geerts, Jianzhong Li, and 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 item sets, 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-item-set mining to reduce the search space. As verified by our experimental study, CFDMiner can be multiple orders of magnitude faster than CTANE and FastCFD for constant CFD discovery. 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-item-set 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 item set, closed item set.

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 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 \( \text{cust} \) 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 \( \text{cust} \) is shown in Fig. 1.

Traditional FDs that hold on \( r_0 \) include the following:

\[
\begin{align*}
  f_1 : \{\text{CC}, \text{AC}\} & \rightarrow \text{CT} \\
  f_2 : \{\text{CC}, \text{AC}, \text{PN}\} & \rightarrow \text{STR}.
\end{align*}
\]

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):
searches an attribute-set containment lattice and derives only constants in both its LHS and RHS. Such CFDs are referred to as FDs with discovery TANE. 1.1 Prior Work

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

In $\phi_0$, $(44, - || -)$ 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 $\phi_1$ 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 || MH); similarly for $\phi_2$ and $\phi_3$. 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 $X$ 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., $\phi_1$-$\phi_3$. 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 $\phi_1$-$\phi_3$ 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.

1.1 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. It 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-item-sets mining (e.g., [23], [24]). With 100 percent confidence, an association rule $(X, t_p) \Rightarrow (A, a)$ is a constant CFD $(X \rightarrow A, (t_p || a))$, 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 nonredundant association rules, which can be found by computing closed item sets and free item sets.

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 data sets with a large arity. 3) A host of techniques have been developed for (nonredundant) association rule mining, and it is only natural to capitalize on these for CFD discovery. As we shall see, these techniques cannot 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.

1.2 Contributions

In the 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 data set \( r \), 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 item sets and free item sets 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 data sets from UCI) and synthetic data sets 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 CFDMiner often outperforms CTANE and FastCFD by three orders of magnitude. We also find that FastCFD scales well with the arity: It is up to three orders of magnitude faster than FastCFD, which scales well with the arity: It is up to three orders of magnitude faster than FastCFD, 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.

1.3 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
In this section, 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.

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

2.1.1 CFDs
A conditional functional dependency (CFD) \( \varphi \) over \( R \) is a pair \((X \rightarrow A, t_p)\), where \( 1) \) \( X \) is a set of attributes in \( \text{attr}(R) \), and \( A \) is a single attribute in \( \text{attr}(R) \), \( 2) \) \( X \rightarrow A \) is a standard FD, referred to as the FD embedded in \( \varphi \), 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 \( \text{dom}(B) \), or an unnamed variable “\( _t \)” that draws values from \( \text{dom}(B) \).

We denote \( X \) as \( \text{LHS}(\varphi) \) and \( A \) as \( \text{RHS}(\varphi) \). If \( A \) also occurs in \( X \), we use \( A_L \) and \( A_R \) to indicate the occurrence of \( A \) in \( \text{LHS}(\varphi) \) and \( \text{RHS}(\varphi) \), respectively. We separate the \( X \) and \( A \) attributes in a pattern tuple with “\( _t \)”.

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

Example 2. The FD \( f_1 \) of Example 1 can be expressed as a CFD \((\{CC, AC\} \rightarrow CT, (_t _t \_t \_t))\); similarly for \( f_2 \). All of \( f_1 \), \( f_2 \), and \( \phi_1 \) are CFDs defined over schema cust. For \( \phi_0 \), for example, \( \text{LHS}(\phi_0) \) is \([CC, ZIP]\) and \( \text{RHS}(\phi_0) \) is STR.

2.1.2 Semantics
To give the semantics of CFDs, we define an order \( \leq \) on constants and the unnamed variable “\( _t \)”: \( \eta_t \leq \eta_t \) if either \( \eta_t = \eta_t \) or \( \eta_t \) is a constant \( a \) and \( \eta_t = \eta_t \).

The order \( \leq \) naturally extends to tuples, e.g., \((44, \text{“EH4 1DT,” “EDI”}) \leq (44, \_\_\_\_) \) but \((01, 07974, \text{“Tree Ave.”}) \not\leq (44, \_\_\_\_) \). 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 \( t_2 \not\geq t_1 \), i.e., when \( t_2 \) is “more general” than \( t_1 \). For instance, \((44, \text{“EH4 1DT,” “EDI”}) \ll (44, \_\_\_\_) \).

An instance \( r \) of \( R \) satisfies the CFD \( \varphi \) (or \( \varphi \) holds on \( r \)), denoted by \( r \models \varphi \), if 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, \( \varphi \) is a constraint defined on the set \( r_{\varphi} = \{ \! \! \! \! t \! \! \! \! | \! \! \! \! r \! \! \! \! \in \! \! \! \! \varphi \! \! \! \! \} \! \! \! \! \} \) such that for any \( t_1, t_2 \in r_{\varphi} \), if \( t_1[X] = t_2[X] \) then \( 1) t_1[A] = t_2[A], \) and \( 2) t_1[A] \leq t_p[A]. \)

Here, \( 1) \) enforces the semantics of the embedded FD on the set \( r_{\varphi} \), and \( 2) \) assures the binding between constants in \( t_p[A] \) and constants in \( t[A] \). That is, \( \varphi \) constrains the subset \( r_{\varphi} \) 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 
\( \phi_0 \phi_3 \) of Example 1. It does not satisfy the CFD \( \psi = \{(CC, ZIP) \rightarrow STR, (c, A) \rightarrow \_\} \). Indeed, \( t_1 \) and \( t_4 \) violate \( \psi \) since 
\( t_1[CC, ZIP] = t_4[CC, ZIP] \leq \_ \), but \( t_1[STR] \neq t_4[STR] \).

Nor does \( r \) satisfy \( \psi' = \{AC \rightarrow CT, (131 \parallel EDI)\} \) since \( t_6 \)
violates \( \psi ' : t_6[AC] \leq (131) \) but \( t_6[CT] \leq \) (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 \( \Sigma \) of CFDs over \( R \), denoted by 
\( r \models \Sigma \), if for each \( \Sigma \) CFD \( \phi \in \Sigma \).

For two sets \( \Sigma \) and \( \Sigma' \) of CFDs defined over the same
schema \( R \), we say that \( \Sigma \) is equivalent to \( \Sigma' \), 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 \((X \rightarrow Y, t_p)\), where \( Y \)
is a set of attributes and \( X \rightarrow Y \) 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.

2.1.3 Classification of CFDs

A CFD \((X \rightarrow A, t_p)\) 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 
\( t_p[A] \neq \_ \), i.e., the RHS of its pattern tuple is the
unnamed variable "."

Example 4. Among the CFDs given in Example 1, \( f_1, f_2, \phi_0 \)
are variable CFDs, while \( \phi_1, \phi_2, \phi_3 \) are constant CFDs.

It has been shown in [1] that any set \( \Sigma \) of CFDs over a
schema \( R \) can be represented by a set \( \Sigma_0 \) of constant CFDs
and a set \( \Sigma_1 \) of variable CFDs, such that \( \Sigma \equiv \Sigma_0 \cup \Sigma_1 \). In
particular, for a CFD \( \delta = (X \rightarrow A, t_p) \), if \( t_p[A] \) is a constant \( a_0 \),
then there is an equivalent CFD \( \delta' = (X' \rightarrow A, (t_p[X'] || a_0)) \),
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 \( \delta \) with \( t_p[B] = "\_" \).

Lemma 1 [1]. For any set \( \Sigma \) of CFDs over a schema \( R \), there exist
a set \( \Sigma_0 \) of constant CFDs and a set \( \Sigma_1 \) of variable CFDs over \( R \),
such that \( \Sigma \) is equivalent to \( \Sigma_0 \cup \Sigma_1 \).

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 nonredundant 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.
2.3 Discovering CFDs with Pattern Tableaus

So far, we considered CFDs of the form \( \varphi = (X \rightarrow A, t_p) \). In [1], however, CFDs were allowed to have \textit{multiple} pattern tuples. More specifically, a tableau CFD is of the form \( \varphi = (X \rightarrow A, T_p) \) where \( T_p \) is a \textit{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 \( \varphi \) 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 \( \{ \varphi_{t_p} | t_p \in T_p \} \). Motivated by this equivalence, we define the support of \( \varphi = (X \rightarrow A, T_p) \), denoted by \( \text{supp}(\varphi) \), as \( \min_{t \in T_p} \text{sup}(\varphi_{t}) \). Consequently, the discovery of \( k \)-frequent tableau CFDs reduces to the problem of discovering \( k \)-frequent CFDs. We, therefore, focus on the latter in this paper.

Furthermore, the notion of minimality can be generalized to tableau CFDs: A CFD \( \varphi = (X \rightarrow A, T_p) \) is minimal on \( r \) if 1) \( r \) \( \not\subset \) \( (Y \rightarrow A, T_p) \) for any proper subset \( Y \subset X \), and 2) the pattern tableau is maximal (it cannot be extended without increasing its support). The equivalence is readily verified (see [1]). Consequently, the discovery of \( k \)-frequent tableau CFDs reduces to the problem of discovering \( k \)-frequent 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 \( (X \rightarrow A, t_p \parallel a) \).

Our algorithm is based on the connection between left-reduced constant CFDs and \textit{free} and \textit{closed} item sets. 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 percent 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 item-set mining.

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

3.1 Free and Closed Item Sets

An item set is a pair \((X, t_p)\), where \( X \subseteq \text{attr}(R) \) and \( t_p \) is a constant pattern over \( X \).

Given an instance \( r \) of the schema \( R \), the \textit{support of \((X, t_p)\)} in \( r \), denoted by \( \text{supp}(X, t_p, r) \), 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 \textit{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 \textit{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 \( \text{supp}(X, t_p, r) \subseteq \text{supp}(Y, s_p, r) \).

An item set \((X, t_p)\) is called \textit{closed} in \( r \) if there exists no item set \((Y, s_p)\) such that \((Y, s_p) \preceq (X, t_p)\) for which \( \text{supp}(Y, s_p, r) = \text{supp}(X, t_p, r) \). Intuitively, a closed item set \((X, t_p)\) cannot be extended without decreasing its support. For an item set \((X, t_p)\), we denote by \( \text{clo}(X, t_p) \) the unique closed item set that extends \((X, t_p)\) and has the same support in \( r \) as \((X, t_p)\).

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

For a natural number \( k \geq 1 \), a closed (respectively free) item set \((X, t_p)\) is called \textit{\( k \)-frequent} if \( |\text{supp}(X, t_p, r)| \geq k \).

Example 6. Fig. 2 shows the closed sets in the \textit{cust} relation (see Fig. 1) that contain \((\text{CT}, \text{MH})\). 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 item sets, but we show the size of the support of the item sets. For example, \((\text{CC}, \text{AC}, \text{CT}, \text{ZIP})\), \((01,908, \text{MH}, 07974)) \) is a closed item set with support equal to 3. This item set has two free patterns, \((\text{CC}, \text{AC})\), \((01,908))\) and \((\text{ZIP})(07974))\), both having support = 3 as well.

The connection between \( k \)-frequent free and closed item sets 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 \( \varphi = (X \rightarrow A, t_p \parallel a) \), \( r \models \varphi \) iff 1) the item set \((X, t_p)\) is free, \( k \)-frequent and it does not contain \((A, a)\), 2) \( \text{clo}(X, t_p) \preceq (A, a) \), and 3) \((X, t_p)\) does not contain a smaller free set \((Y, s_p)\) with this property, i.e., there exists no \((Y, s_p)\) such that \((X, t_p) \preceq (Y, s_p)\), \( Y \subseteq X \), and \( \text{clo}(Y, s_p) \preceq (A, a) \).

Example 7. From Proposition 1 and the closed and free item sets shown in Fig. 2, it follows that \( \phi_1 \) (\((\text{CC}, \text{AC}) \rightarrow \text{CT}, (01,908)\)) of Example 1 is a 3-frequent constant CFD that holds on the \textit{cust} relation. Indeed, it is obtained from the closed pattern \((\text{CC}, \text{AC}, \text{CT}, \text{ZIP})\), \((01,908, \text{MH}, 07974))\), where the free pattern \((\text{CC}, \text{AC})\), \((01,908))\) is taken as the LHS of the constant CFD. Fig. 2, however, shows that this LHS contains a smaller free set \((\text{AC}, (908))\) whose closed set \((\text{AC}, \text{CT})\), \((908, \text{MH}))\) contains \((\text{CT}, \text{MH})\). Hence, \( \phi_1 \) is not left-reduced. It is easily verified that \((\text{AC} \rightarrow \text{CT}, (908))\) is a 4-frequent left-reduced constant CFD on \textit{cust}. Similarly \( \phi_2 \) of Example 1 can be obtained (although one has to consider closed patterns that contain \((\text{CT}, \text{ED1}))\) for \( \phi_2 \).
3.2 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 \[26\]. We opt for the GCGROWTH algorithm of \[26\] 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 \[26\] for more details. For our purposes, it is sufficient to know that GCGROWTH returns a mapping \( C^2F \) that associates with each \( k \)-frequent closed item set its set of \( k \)-frequent free item sets.

Given this mapping, CFDMiner works as follows:

1. For each \( k \)-frequent closed item set \((X, t_p)\) we add its free sets, as given by \( C^2F \), to a hash table \( H \).
2. For each closed item set \((X, t_p)\), we associate with each of its free item sets \((Y, s_p)\) the item set \( \text{RHS}(Y, s_p) = (X \setminus Y, t_p[X \setminus Y]) \). 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 item sets is constructed as well. Item sets in this list are sorted in ascending order \( w.r.t. \) their sizes.
3. For each free item set \((Y, s_p)\) in the list \( L \), CFDMiner does the following:
   a. For each subset \( Y' \subseteq Y \) 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 is not already included in some \( \text{RHS}(Y', s_p[Y']) \) of one of its sub-item sets. 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 \((Y \rightarrow A, (s_p \parallel a))\) 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 \( L \) in a levelwise way. More precisely, the lattice \( L \) consists of elements of the form \((X, t_p)\), where \( X \subseteq \text{attr}(R) \) and \( t_p \) is pattern tuple over \( X \). In contrast to the item sets 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 \((A, a)\) for \( A \in \text{attr}(R) \) and \( a \in \text{dom}(A) \cup \{\_\} \). It then proceeds to larger attribute-set/pattern levels in \( L \). When it inspects \((X, s_p)\), it checks CFDs \((X \setminus \{A\} \rightarrow A, (s_p[X \setminus \{A\}] \parallel s_p[A])\), where \( A \subseteq X \). This guarantees that only nontrivial CFDs are considered. Furthermore, CTANE maintains for each considered element \((X, s_p)\) a set, denoted by \( C^+(X, s_p) \), to determine whether CFD \((X \setminus \{A\} \rightarrow A, (s_p[X \setminus \{A\}] \parallel s_p[A])\) is minimal. The set \( C^+(X, s_p) \), as will be elaborated below, can be maintained during the levelwise traversal. Apart from testing for minimality, \( C^+(X, s_p) \) also provides an effective pruning strategy, making the levelwise approach feasible in practice.

4.1 Pruning Strategy

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

1. If \( A \subseteq X \), then \( c_A = s_p[A] \).
2. \( r \not\subseteq (X \setminus \{A, B\} \rightarrow B, (s_p[X \setminus \{A, B\}] \parallel s_p[B])) \) for all \( B \subseteq X \), and
3. For all \( B \subseteq X \setminus \{A\} \), \( r \not\subseteq (X \setminus \{A\} \rightarrow A, (s^B_p \parallel c_A) \), where \( s^B_p[C] = s_p[C] \) for all \( C \neq B \) and \( s^B_p[B] = _\_ \).

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 \( X \subseteq \text{attr}(R) \), \( s_p \) be a pattern over \( X \), \( A \subseteq X \) and assume that \( r \not\subseteq \varphi = (X \setminus \{A\} \rightarrow A, (s_p[X \setminus \{A\}] \parallel s_p[A])) \).

Then \( \varphi \) is minimal iff for all \( B \subseteq X \) we have that \((A, s_p[A]) \in C^+(X \setminus \{B\}, s_p[X \setminus \{B\}]) \).

In terms of pruning, Lemma 2 says that we do not need to consider any element \((X, s_p)\) of \( L \) for which \( C^+(X, s_p) = \emptyset \).

Moreover, if \( C^+(X, s_p) = \emptyset \) then also \( C^+(Y, t_p) = \emptyset \) for any \((Y, s_p)\) that contains \((X, t_p)\) in the lattice. Therefore, the emptiness of \( C^+(X, s_p) \) potentially prunes away a large part of elements in \( L \) that otherwise need to be considered by CTANE.

Algorithm CTANE. We are now ready to present the algorithm. We denote by \( L_\ell \) a collection of elements \((X, s_p)\) in \( L \) of size \( \ell \), i.e., \( |X| = \ell \). We assume that \( L_\ell \) is ordered such that \((X, s_p)\) appears before \((Y, t_p)\) if \( X = Y \) and \( t_p \ll s_p \).

Initially, \( L_1 = \{(A, _\_) | A \in \text{attr}(R)\} \cup \{(A, a) | a \in \pi_A(r), A \in \text{attr}(R)\}, C^+(\emptyset) = L_1 \) and \( \ell = 1 \). We then execute the following steps as long as \( L_\ell \) is nonempty:

1. We compute candidate RHS for minimal CFDs with their LHS in \( L_\ell \). That is, for each \((X, s_p) \in L_\ell \) we compute
   \[ C^+(X, s_p) = \bigcap_{B \subseteq X} C^+(X \setminus \{B\}, s_p[X \setminus \{B\}]) \]

2. For each \((X, s_p) \in L_\ell \) we look for valid CFDs; i.e., for each \( A \subseteq X \), \((A, c_A) \in C^+(X, s_p) \) we do the following:
a. check whether \( r \models \varphi \), where
\[
\varphi = (X \setminus \{A\} \rightarrow A, (s_p[X \setminus \{A\}] \| c_A)),
\]

b. if \( r \models \varphi \) then output \( \varphi \). Indeed, if \( \varphi \) holds on \( r \) then by Lemma 2 and Step 1, \( \varphi \) is indeed a minimal CFD,

c. if \( r \models \varphi \) then for all \((X, u_p) \in L_t\) such that
\[
\begin{align*}
&u_p[A] = c_A \quad \text{and} \quad u_p[X \setminus \{A\}] \leq s_p[X \setminus \{A\}],

&\text{update } C^+(X, u_p) \text{ by removing } (A, c_A) \text{ and } (B, c_B) \text{ from it, for all } B \in \text{attr}(R) \setminus X.
\end{align*}
\]

3. Next, we prune \( L_t \). That is, for each \((X, s_p) \in L_t\) we remove \((X, s_p)\) from \( L_t \) provided that \( C^+(X, s_p) = \emptyset \).

4. Finally, we generate \( L_{t+1} \) as follows:

a. initially \( L_{t+1} = \emptyset \),

b. for each pair of distinct \((X, s_p), (Y, t_p) \in L_t\) 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 \( \pi_Z(r) \) matches \( u_p \) then continue with \((Z, u_p)\),

iii. if for all \( A \in Z \), \((Z \setminus \{A\}, u_p[Z \setminus \{A\}]) \in L_t\), then add \((Z, u_p)\) to \( L_{t+1} \),

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.

### 4.2 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 \( \varphi = (X \rightarrow A, (t_p, c_A)) \) be a CFD that holds on \( r \). We denote by \((X^c, t_p^c)\) the item set consisting of the constant part of \((X, t_p)\). Then \( \varphi \) is \( k\)-frequent iff \( \text{supp}(X^c, t_p^c, r) \geq k \) 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 \((X, s_p) \in L_t\) to those for which \((X^c, s_p^c)\) is a \( k\)-frequent item-set. This can be achieved by (1) starting with \( L_1 = \{(A, \_ | A \in \text{attr}(R)) \cup \{(A, a_1) | \text{supp}(A, a_1, r) \geq k, A \in \text{attr}(R))\}; and 2) replacing Step 4.b(ii) in CTANE by a step that only considers \((Z, u_p)\) if \( \text{supp}(Z^c, u_p^c, r) \geq k \). 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 attribute 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: \((ZIP \rightarrow CC, (07974 || _))\), \((ZIP \rightarrow CC, (07974 || 01))\), \((ZIP \rightarrow AC, (07974 || _))\), \((ZIP \rightarrow AC, (07974 || 0988))\), and \((STR \rightarrow ZIP, (_ || _))\). 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 \((CC, AC) \rightarrow ZIP, (_, _))\).

(E) As a result, CTANE updates the \( C^+\)-sets in Step 2.c, not only of the current pattern 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 CFDs 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 \((AC, CC) \rightarrow ZIP, (_, _ || 07974))\), 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.
4.3 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 $C^+(X, sp)$:

**Lemma 3.** Suppose that for all $(Y, t_p) \in L_{\ell_1}$, $C^+(Y, t_p)$ is correctly computed. Then Steps 1 and 2.c of CTANE correctly compute $C^+(X, sp)$ for all $(X, sp) \in L_{\ell_1}$.

4.4 Implementation Details
We now briefly elaborate on the implementation of FastCFD:

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 $\varphi$ such that $sup(\varphi, r) \geq k$. 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 right-hand side attribute. More specifically, for each attribute $A$ in $attr(R)$, FastCFD looks for all CFDs of the form $\varphi = (Y \rightarrow A, t_p)$ such that $Y \subseteq attr(R) \setminus \{A\}$, $\varphi$ is minimal, and moreover $sup(\varphi, r) \geq k$. We denote this set of CFDs by $Cover(A, r, k)$. Clearly, all $k$-frequent minimal CFDs in $r$ can then be obtained as $\bigcup_{A \in attr(R)} Cover(A, r, k)$. The technical challenge of FastCFD, therefore, shifts to the computation of $Cover(A, r, k)$ for a given $A \in attr(R)$, $r$ and $k \geq 0$.

It is to compute $Cover(A, r, k)$ that FastCFD leverages a depth-first search strategy. More specifically, the key observation behind FastCFD is a relationship between CFDs $\varphi = (Y \rightarrow A, t_p)$ in $Cover(A, r, k)$ 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 possibly 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 $Cover(A, r, k)$.

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 is sufficient to consider free item sets only). For each $(X, t_p)$, it invokes a recursive procedure, denoted by FindMin. This procedure will extend $X$ by all subsets $Y$ in $attr(R) \setminus X \cup \{A\}$, and test whether the resulting CFD $(X, Y) \rightarrow (t_p, t_0, \ldots, t_k)$ 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 $attr(R) \setminus X \cup \{A\}$.

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

5.1 Difference Sets
As previously mentioned, to compute $Cover(A, r, k)$ 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

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

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

We denote by $D_{A}(r)$ the set $\{Y \setminus \{A\} \mid Y \in D(r), A \in Y\}$, 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 $Y \in D_{A}(r)$ is said to be minimal if for all $Y' \in D_{A}(r)$ such that $Y' \subseteq Y$ then $Y' = Y$. We denote the set of minimal difference sets in $D_{A}(r)$ by $D'_{A}(r)$.
To characterize the relationship between minimal difference sets of minimal CFDs in $\text{Cover}(A, r, k)$, we need the following notations. Denote by $P(\text{attr}(R))$ the power set of $\text{attr}(R)$. Let $Z \subseteq \text{attr}(R)$ and $X \subseteq P(\text{attr}(R))$. 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.**

1. For any constant CFD $\phi = (X \rightarrow A, (t_p \parallel a), r \models \phi$ and $\sup(\phi, r) \geq k$ iff $|r_{t_p}| \geq k$, $D^n_A(r_{t_p}) = \emptyset$, and $\pi_A(r_{t_p}) = (a)$.
2. For any variable CFD $\phi = (X \rightarrow A, (t_p \parallel \_), r \models \phi$ and $\sup(\phi, r) \geq k$ iff $|r_{t_p}| \geq k$ and $X$ covers $D^n_A(r_{t_p})$.

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

Lemma 4 provides 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 $(X \rightarrow A, (t_p \parallel a))$, Lemma 4 tells us that we need to find a $k$-frequent item set $(X, t_p)$ in $r$, such that $D^n_A(r_{t_p}) = \emptyset$ and $D^n_A(r_{t_p}[X^\perp]) \neq \emptyset$ for any $X^\perp \subseteq X$ 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 $(X \rightarrow A, (t_p \parallel \_))$. Observe that sets in $D^n_A(r_{t_p})$ 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 $[(X^C, X^V) \rightarrow A, (t_p^C, t_p^V \parallel \_)]$ we have to find a $k$-frequent item set $(X^C, t_p^C)$ in $r$ such that

- $X^C$ is a minimal cover of $D^n_A(r_{t_p^C})$, i.e., there exists no $Y^C \subseteq X^C$ of size $|X^C| - 1$ that covers $D^n_A(r_{t_p^C})$.
- none of the constants in $t_p^C$ can be replaced by a “$\_”

i.e., there exists no $X^C \subseteq X^C$ of size $|X^C| - 1$ such that $X^C \cup (X \setminus X^C)$ covers $D^n_A(r_{t_p[X \setminus X^C]})$.

Conditions (b1) and (b2) are on $X^C$ and $X^V$, 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 $\text{attr}(R) \setminus \{A\}$ to find minimal covers of the difference sets $D^n_A(r_{t_p})$ 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.

### 5.2 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 item set. Similar to the constant CFD case (see Proposition 1) we now show that it is not necessary to consider all $k$-frequent item sets $(X^C, t_p^C)$ when discovering minimal variable CFDs.

Indeed, the following lemma tells us that it suffices to consider only $k$-frequent free item sets. This yields a pruning strategy, i.e., by only considering free item sets. As we will see 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 $\phi$ is minimal then the constant pattern in $t_p$, denoted by $(X^C, t_p^C)$, is a $k$-frequent free item set.

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

**Algorithm FindCover.** Procedure FindCover($A, r, k$), in turn, invokes the recursive procedure FindMin. More specifically, Proposition 1 and Lemma 5 state that it is sufficient to consider $k$-frequent free item sets as constant patterns of CFDs only. Hence, FindCover first extracts the set of the $k$-frequent free item sets $\text{Fr}(r)$ of $r$, in which item sets are kept in the ascending order $\text{w.r.t.}$ their sizes. To efficiently retrieve elements in $\text{Fr}(r)$, FindCover also indexes those item sets in a hash table.

Second, for each item set $(X, t_p) \in \text{Fr}(r)$, FindCover maintains $D^n_A(r_{t_p})$, i.e., the set of minimal difference sets produced from all tuples in $r_{t_p}$. Then, for a given $(X, t_p) \in \text{Fr}(r)$, FindCover recursively calls FindMin to find a minimal cover $Y$ of $D^n_A(r_{t_p})$ and tests conditions (a), (b1), and (b2), previously described.

**Algorithm FindMin.** Procedure FindMin finds the minimal covers by traversing all subsets of $\text{attr}(R) \setminus \{A\}$ in a depth-first fashion. That is, we assume an ordering $<_{\text{attr}}$ on $\text{attr}(R)$. All subsets of $\text{attr}(R) \setminus \{A\}$ are then enumerated in a depth-first, left-to-right fashion based on the given attribute ordering. For instance, suppose that $\text{attr}(R) = \{A, B, C, D\}$ and $A <_{\text{attr}} B <_{\text{attr}} C <_{\text{attr}} D$. Then, starting from the empty set, the subsets of $\text{attr}(R) \setminus \{A\}$ 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 $<_{\text{attr}}$, 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 $Y \subseteq \text{attr}(R)$ and its corresponding path in the tree by $Y$.

During the enumeration of the subsets by FindMin, we denote by $Y \subseteq \text{attr}(R)$ the current path in the enumeration tree. Furthermore, when inspecting $Y$, FindMin maintains the difference sets in $D^n_A(r_{t_p})$ that are currently not covered yet by attributes in $Y$. We denote this set by $D^n_A(r_{t_p})[Y]$. Initially, i.e., when $Y = \emptyset$, this set is equal to $D^n_A(r_{t_p})$. The details of FindMin are as follows:
Input: $A \in \text{attr}(R)$, $(X, t_p) \in \text{Fr}_x(r)$, $Y \subseteq \text{attr}(R) \setminus \{A\}$, $D^\text{STR}_A(r_p)[Y]$, and $\prec_{\text{attr}}$.

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

Base case:
1) If $\emptyset \in D^\text{STR}_A(r_p)[Y]$, then return an empty set. By Lemma 4, $(\{X, Y\}, (t_p, \ldots, \parallel t_o))$ can never lead to a valid CFD.

2) If $Y$ contains the last attributes in $\text{attr}(R) \setminus \{A\}$ w.r.t. $\prec_{\text{attr}}$, but $D^\text{STR}_A(r_p)[Y] \neq \emptyset$, then return an empty set. By Lemma 4, $r \neq ([X, Y] \rightarrow A, (t_p, \ldots, \parallel \lambda)))$ because $Y$ does not cover $D^\text{STR}_A(r_p)$; moreover, since $([X, Y], (t_p, \ldots, \parallel \lambda))$ cannot be further extended, this pattern does not lead to a valid CFD.

3) If $D^\text{STR}_A(r_p)[Y] = \emptyset$, then $Y$ is a cover of $D^\text{STR}_A(r_p)$. There are two cases to consider corresponding to the conditions (a) and (b1-b2).

   a) If $D^\text{STR}_A(r_p) = \emptyset$, then by Lemma 4, there exists a constant $t_o$, $r \models (X \rightarrow A, (t_p \parallel t_o))$. In order to check for minimality, we need to verify whether there is no $X' \subseteq X$ of size $|X| - 1$ such that $r \models (X' \rightarrow A, (t_p[\lambda] \parallel t_o))$. If this holds, then output constant CFD $(X \rightarrow A, (t_p \parallel t_o))$.

   b) If $D^\text{STR}_A(r_p) \neq \emptyset$, then Lemma 4 implies that $r \models ([X, Y] \rightarrow A, (t_p, \ldots, \parallel \lambda)))$. In order to check for minimality, we need to verify whether:
   i) there is no $Y' \subseteq Y$ of size $|Y| - 1$ such that $Y'$ covers $D^\text{STR}_A(r_p[X])$;
   ii) there is no $X' \subseteq X$ of size $|X| - 1$ such that $Y \cup (X' \setminus X)$ covers $D^\text{STR}_A(r_p[X'])$.

   If conditions 1) and 2) are both satisfied, then output variable CFD $([X, Y] \rightarrow A, (t_p, \ldots, \parallel \lambda))$.

Recursive case:
4) For each attribute $B$ that appears after $Y$ w.r.t. $\prec_{\text{attr}}$, we do the following:
   a) Let $Y' = Y \cup \{B\}$ and $D^\text{STR}_A(r_p)[Y']$ be the difference sets of $D^\text{STR}_A(r_p[Y])$ not covered by $B$.
   b) Call FindMin($A, (X, t_p), Y', D^\text{STR}_A(r_p)[Y'], \prec_{\text{attr}}$) recursively following the depth-first strategy.

Before illustrating algorithm FastCFD, we remark the following. A careful reader might wonder how $D^\text{STR}_A(r_p[X])$ is obtained in Step 3b(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 item set due to the anti-monotonicity property of frequent item sets. Furthermore, there exist k-frequent free item sets $(Z, s_p)$ such that $(Z, s_p) \preceq (X, t_p[X])$. Because $\text{sup}[X, t_p[X]]) = \max \{\text{sup}[Z, s_p] \}$, $D^\text{STR}_A(r_p[X])$ is the same as $D^\text{STR}_A(r_p[z])$, where $(Z, s_p)$ is the free item set with the maximum cardinality for all $(Z, s_p) \preceq (X, t_p[X])$. Since $D^\text{STR}_A(r_p[z])$ is maintained, we use this set in Step 3b(ii).

Example 9. Consider again the cust relation of Fig. 1. We give a partial run of FindCover($\text{attr}(R) \setminus \text{STR, STR, cust, 2})$ 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 $\prec_{\text{attr}}$ orders attributes alphabetically. Fig. 4 depicts the following: in the top right corner we have shown a partial set of the $k$-frequent free item sets in $\text{Fr}_x(r)$, where the immediate subsets of the free item sets are illustrated by dotted arrows; in the bottom left corner we have drawn a partial execution tree of FindMin for the free pattern (CC, 01). We highlight circled points $A, B, C$ and $D$ during the execution of FindCover. Note that the execution of FindCover constructs a depth-first search tree for every free pattern although only the one for (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 $(Y \rightarrow \text{STR}, (\ldots, \parallel \lambda))$ are returned.

(B) Next, we consider the free pattern (CC, 01) with $r_{CC=01} = \{t_1, t_2, t_3, t_4, t_5\}$, and minimal difference sets $D^\text{STR}_A(r_{CC=01}) = \{[PN], [AC, CT]\}$.

Hence, FindMin will find a cover for $D^\text{STR}_A(r_{CC=01})$ by the recursive process invoked in Step 4. The (partial) enumeration tree of the subsets of $\{AC, CT, PN\}$ according to $\prec_{\text{attr}}$ is illustrated in Fig. 4. The corresponding covers $Y$ of $D^\text{STR}_A(r_{CC=01})$ computed are $[AC, PN]$ and $[CT, PN]$. Consider the cover $[AC, PN]$ and its minimal CFD candidate $\phi' = ([AC, AC, PN] \rightarrow \text{STR}, (01, \ldots, \parallel \lambda))$ in Step 3b. Although the algorithm verifies that $\phi'$ is minimal for $r_{CC=01}$ in Step 3b(ii), it still needs to inspect whether $[CC, AC, PN]$ covers $D^\text{STR}_A(r_0)$ in Step 3b(iii), where $\emptyset$ is the only immediate subset of pattern (CC, 01). In this case, it finds out that $[CC, AC, PN]$ covers $D^\text{STR}_A(r_0)$ in Step 3b(iii), which indicates that $r \models ([CC, AC, PN] \rightarrow \text{STR}, (\ldots, \parallel \lambda))$. Thus, $\phi'$ is not a minimal CFD.

Similarly, consider the free pattern (CC, 44) with $r_{CC=44} = \{t_5, t_6, t_7\}$ and minimal difference sets $D^\text{STR}_A(r_{CC=44}) = \{[AC, CT, ZIP]\}$.
The covers of $D_{\text{STR}}^{m}(r_{\text{CC-44}})$ are AC, CT, and ZIP. For the cover AC, FindMin needs to inspect if its CFD

$$\phi = ([\text{CC}, \text{AC}] \rightarrow \text{STR}, (44, \|, \|))$$

is minimal. In Step 3.b(i), it verifies that $\phi$ is minimal for $r_{\text{CC-44}}$, but it still needs to inspect whether $[\text{CC}, \text{AC}]$ covers $D_{\text{STR}}^{m}(r_{b})$ (i.e., $D_{\text{STR}}^{m}(r)$) in Step 3.b(ii), where again $\emptyset$ is the only immediate subset of pattern (CC, 44). As we can observe from the cost relation, $D(t_{1}, t_{2}) = ([\text{PN}, \text{STR}], \text{AC})$, and $[\text{PN}] \in D_{\text{STR}}^{m}(r)$ (one may compute $D_{\text{STR}}^{m}(r)$ to verify that $[\text{PN}] \in D_{\text{STR}}^{m}(r)$). This implies that $[\text{CC}, \text{AC}]$ cannot be a cover for $D_{\text{STR}}^{m}(r)$. Thus, $\phi$ is a minimal CFD.

**D** As a final example, we consider the free set $(X, t_{p}) = ([\text{CC}, \text{AC}], [01, 908])$ with $r_{t} = \{t_{1}, t_{2}, t_{4}\}$ and minimal difference sets

$$D_{\text{STR}}^{m}(r_{t}) = \{[\text{PN}]\}.$$

The corresponding cover of $D_{\text{STR}}^{m}(r_{t})$ is $[\text{PN}]$. Consider its minimal CFD candidate

$$\phi^{*} = ([\text{CC}, \text{AC}, \text{PN}] \rightarrow \text{STR}, (01, 908, \|, \|))$$

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

### 5.3 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 $\phi$ in terms of the minimality of the constant pattern and unnamed variables in LHS($\phi$), and 2) the fact that we discover $k$-frequent CFDs instead of 1-frequent FDs only. Whereas forFDs, the only difference sets needed are $D_{A}^{m}(r)$ for $A \in \text{attr}(R)$. Lemma 4 states that for CFDs, difference sets $D_{A}^{m}(r_{t})$ are needed for all $r_{t}$, where $t_{0}$ is a $k$-frequent free pattern in $r$. Worse still, when $(X, t_{p})$ is reached, the depth-first approach enforces FindMin to use $D_{A}^{m}(r_{t_{0}})$ 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.

### 5.4 NaiveFast

The first one is inspired by the stripped partition-based approach used by FastFD [14]. Here, for a given $(X, t_{p})$ the stripped partition of $r_{t_{0}}$ w.r.t. an attribute $A$ is the partition of $r_{t_{0}}$ 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 $r_{t}$, for each $A \in \text{attr}(R)$ basically 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 is important to remark 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.

### 5.5 FastCFD

The second approach relies on the availability of $\text{Closed}_{2}(r)$, which consists of all 2-frequent closed item sets in $r$. Given $(X, t_{p})$, we can again infer for any two tuples in $r_{t}$ on which attributes they agree. Indeed, these sets of attributes are given by the attributes in those item sets in $\text{Closed}_{2}(r)$ that match $t_{p}$ (the constant part of $t_{p}$). By taking the complement we can infer the desired difference sets efficiently. Our experimental evaluation (see Section 6) shows that this approach outperforms the partition-based approach, and is, therefore, taken as the default implementation for difference sets in FastCFD.

Finally, since CFDMiner produces $\text{Closed}_{2}(r)$ as a side-product, we can choose to use CFDMiner for constant CFD discovery and use 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 reported in the next section.

### 5.6 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 $<_{\text{attr}}$ fixed throughout the execution of FindMin, an additional step (between Steps 4.a and 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, and 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:

1. the support threshold $k$,
2. the size DBSIZE of a sample relation $r$, i.e., the number of tuples in $r$,
3. the arity ARITY of $r$, i.e., the number of columns in $r$,
4. a correlation factor (CF) [14], which indicates that the average range of distinct values in an attribute domain is $\text{CF} \times \text{DBSIZE}$.

#### 6.1 Experimental Settings

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Arity</th>
<th>Size (# of tuples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wisconsin breast cancer (WBC)</td>
<td>11</td>
<td>699</td>
</tr>
<tr>
<td>Chess</td>
<td>7</td>
<td>28,056</td>
</tr>
<tr>
<td>Tax</td>
<td>14</td>
<td>20,000</td>
</tr>
</tbody>
</table>

To evaluate the scalability of the algorithms, we also used an extension of the relation in Fig. 1, which is a synthetic data set 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 data sets accordingly.

The algorithms have been implemented in C++. The program has been tested on AMD Opteron Processor (2.6GHz) with 32 GB of memory running Linux operating system. Our algorithms run entirely in main memory. Each experiment was repeated over five 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.

6.2.1 Scalability Experiments

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 20\( K \) to 1 million tuples. We kept support ratio SUP percent, which is defined as \( \frac{k}{\text{DBSIZE}} \) at 0.1 percent. 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. 1) CFDMiner, which only mines constant CFDs, is multiple orders of magnitude faster than the other algorithms that discover both constant and variable CFDs. 2) The naïve 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 100 \( K \). But when DBSIZE is 300 \( K \), 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 FastTFD [14], the difference set construction contributes most to the cost of NaiveFast. When DBSIZE becomes larger, there are more item sets with large support that need to be considered for constructing the difference sets. This results in a significant performance degradation of NaiveFast. 3) 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-item sets from CFDMiner for constructing difference sets.

Fig. 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 CF = 0.7, DBSIZE = 20 \( K \), and 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 as ARITY increases. Both are orders of magnitude better than CTANE when ARITY \( \geq 15 \). In addition, FastCFD is four times better than NaiveFast when ARITY reaches 31, which further demonstrates the effectiveness of the optimization techniques of FastCFD via CFDMiner.

**Scalability w.r.t. k.** We fixed CF = 0.7, DBSIZE = 100 \( K \), 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 item sets with large support satisfy \( k \) when \( k \) becomes larger, 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 item sets with large support, which does not change significantly when \( k \) gets larger.

Fig. 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 DBSIZE = 50 K, k = 50 and 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 item sets with large support increases. For a fixed k, this means more item sets 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.

6.2.2 Real Data Experiments
We have conducted experiments on real-life data, including the Chess, WBC, and synthetic Tax data sets. For each data set, k was varied. Figs. 11, 12, and 13 show the response times of CTANE and FastCFD when k is varied, while Figs. 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.

6.2.3 Summary
From the experimental results, we find the following:

1. CFDMiner can be multiple orders of magnitude faster than CTANE and FastCFD for constant CFD discovery.
2. **CTANE** usually works well when the arity of a sample relation is small and the support threshold is large, but it scales poorly when the arity of a relation increases.

3. **NaiveFast** and **FastCFD** are far more efficient than **CTANE** when the arity of the relation is large.

4. Our optimization technique based on closed-item-set 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 [27], propagation of CFDs from source data to views in data integration [28], extensions of CFDs by adding disjunction and negation [29] or adding ranges [10], confidence of CFDs [30], as well as extensions of inclusion dependencies with conditions (referred to as CINDs) [31]. To our knowledge, CFD discovery was only studied in [21], [10], [32]. 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], [32]. 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], [32]. 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 $\chi^2$-test. The algorithm of [21] differs from our algorithms in the following aspects: 1) 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 data sets. It is to deal with such data that **FastCFD** is provided. In contrast, this issue is not addressed in [21]. 2) In addition, constant CFD discovery is not considered in [21], despite its wide applications in data cleaning and data integration. 3) Moreover, [21] does not consider optimizations based on closed-item-sets mining, which is employed by **FastCFD**.

As observed in Sections 1 and 3, CFD discovery is also closely related to (nonredundant) association rule mining [22], [23], [24]. In particular, CFDMiner is based on the mining algorithm proposed in [24]. Recently, an algorithm was proposed in [32] 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 [32] may be used to mine general CFDs and CINDs. The method of [32], however, can only discover CFDs with 100 percent confidence, and in addition, the minimality of CFDs is not investigated in [32]. 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 [25]. Neither [32] nor [25] 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-item-set 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 data set 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 data sets collected from real life. Second, we are investigating how to discover minimal CFDs from a data set 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 item sets, 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

Wenfei Fan received the BS and MS degrees from Peking University, and the PhD degree from the University of Pennsylvania. He is a professor in the School of Informatics, University of Edinburgh, United Kingdom. He is a recipient of the Alberto O. Mendelzon Test-of-Time Award of ACM PODS 2010, the Roger Needham Award 2008, the Outstanding Overseas Young Scholar Award 2003, the Career Award 2001, the ICDE Best Paper Award 2007, and the Best Paper of the Year Award 2002 from Computer Networks. His current research interests include data quality, data integration, distributed query processing, web services, and XML.

Floris Geerts received the PhD degree from Hasselt University, Belgium. Currently, he is a senior research fellow at the University of Edinburgh. His research interests include data quality, annotation management, and database query languages.

Jianzhong Li is a professor in the School of Computer Science and Technology, Harbin Institute of Technology, China. He worked at the University of California at Berkeley as a visiting scholar in 1985. From 1986 to 1987 and 1992 to 1993, he was a staff scientist in the Information Research Group at Lawrence Berkeley National Laboratory, Berkeley. His current research interests include database management systems, data warehousing, data mining, sensor network, and data intensive super computing. He has authored three books and has published more than 200 research papers.

Ming Xiong received the BS degree in computer science and engineering from Xian Jiaotong University, MS degree in computer science from Sichuan University, and the PhD degree in computer science from University of Massachusetts, Amherst. From 2000 to 2009, he was a member of the Technical Staff at Bell Laboratories Research, Lucent Technologies. His research interests include real-time systems, database systems, and mobile computing. He is currently a member of the Technical Staff at Google Inc.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.