Characterizing Covers of Functional Dependencies using FCA

Victor Codocedo[1], Jaume Baixeries[2], Mehdi Kaytoue[3], and Amedeo Napoli[4]

[1] Universidad Técnica Federico Santa María. Campus San Joaquín, Santiago de Chile.
[2] Universitat Politècnica de Catalunya. 08032, Barcelona. Catalonia.
[3] Infologic, 99 Avenue de Lyon, F-26500, Bourg-lès-Valence, France
[4] LORIA (CNRS - Inria Nancy Grand Est - Université de Lorraine), B.P. 239, F-54506, Vandœuvre-lès-Nancy.

Corresponding author: victor.codocedo@inf.utfsm.cl

Abstract. Functional dependencies (FDs) can be used for various important operations on data, for instance, checking the consistency and the quality of a database (including databases that contain complex data). Consequently, a generic framework that allows mining a sound, complete, non-redundant and yet compact set of FDs is an important tool for many different applications. There are different definitions of such sets of FDs (usually called *covers*).
In this paper, we present the characterization of two different kinds of covers for FDs in terms of pattern structures. The convenience of such a characterization is that it allows for an easy implementation of efficient mining algorithms which can later be easily adapted to other kinds of similar dependencies. Finally, we present empirical evidence that the proposed approach can perform better than a state-of-the-art FD miner in large databases.

1 Introduction

Functional Dependencies (FDs) are a keystone of the relational database model, since they allow checking the consistency, maintaining the quality of a database [8, 10, 9], and guiding its design [20]. In addition, they have been used to study information integration in the Web of data with varying degrees of quality [24, 25], or to check the data completeness in DBpedia [1]. Therefore, the computation of a succinct representation of a set of FDs (usually refered to as a **cover**), is of interest to various fields of knowledge discovery and representation, specially, if this computation can easily be extended to other kinds of dependencies (e.g. relaxed versions of FDs [5]).

The computation of FD *covers* is a popular topic in the database literature. As a reference, in [21], seven algorithms to mine FDs and compute their covers are reviewed and grouped into three families: *lattice transversal algorithms*, *difference/agree sets algorithms*, and *dependency induction* algorithms. The characterization of FDs with FCA and pattern structures is also presented in [2] and

a generalization to other types of FDs is given in [6]. For a detailed review on the characterization of FDs and FCA, see [3].

In this paper, we characterize the representations of FD covers using pattern structures, an extension of FCA dealing with complex object representations [18].

On the one hand, we adapt the definition of a canonical direct basis of implications with proper premises [4, 22] to the formalism of pattern structures, in order to prove that this basis is equivalent to a reduced non-redundant set of FDs, better known as the canonical cover (Section 3.1). We show that the canonical cover can be characterized using the arrow relation ($\swarrow$) between an attribute and a pattern defined over a partition pattern structures (PPS). On the other hand, we discuss on the relation between the Stem Base of implications [12] with the Minimal Cover of dependencies, a sound, complete, non-redundant set of FDs that has minimum cardinality w.r.t. any other equivalent cover (Section 3.2). We show that the latter can be characterized by pseudo-extents of a PPS.

Finally, in Section 4 we empirically compare these two ways of computing FD covers with the algorithm TANE [16]. This algorithm is one of the most efficient FD mining algorithms and according to [21], it is the base for the family of "lattice transversal algorithms" serving as the baseline to validate our approach with a state-of-the-art FD miner.

2 Theoretical Background

Let $\mathcal{U}$ be an ordered set of attributes, and let Dom be a set of values (a domain). For the sake of simplicity, we assume that Dom is a numerical set. A tuple t is a function $t : \mathcal{U} \to Dom$, and a table T is a set of tuples. Usually a table is represented as a matrix, as in Table 1, where the set of tuples (or objects) is $T = \{ t_1, t_2, \ldots, t_7 \}$ with attributes $\mathcal{U} = \{ a, b, c, d, e \}$. We use *table, dataset, set of tuples* as equivalent terms. We overload the functional notation of a tuple in such a way that, given a tuple $t \in T$, we say that $t(X \subseteq \mathcal{U})$ is a tuple with the values of t in the ordered attributes $x_i \in X$ defined as $t(X) = \langle t(x_1), t(x_2), \ldots, t(x_n) \rangle$. For example, we have that $t_2(\{ a, c \}) = \langle t_2(a), t_2(c) \rangle = \langle 2, 1 \rangle$. In this article, the set notation is dropped: instead of $\{ a, b \}$ we use ab.

2.1 Functional Dependencies and their Covers

Definition 1 ([23]). *Let T be a set of tuples, and $X, Y \subseteq \mathcal{U}$. A* **functional dependency (FD)** *$X \to Y$ holds in T if:*

$$\forall t, t' \in T : t(X) = t'(X) \implies t(Y) = t'(Y)$$

For instance, the functional dependency $d \to e$ holds in T (Table 1), whereas the functional dependency $e \to d$ does not hold since $t_4(e) = t_5(e)$ but $t_4(d) \neq t_5(d)$.

For a given set of functional dependencies F, we can use the three Armstrong's axioms (reflexivity, augmentation and transitivity) to derive a larger

set of FDs [20]. We will call F^* the set of FDs derived from F by reflexivity and augmentation, and F^+ the set of FDs derived by reflexivity, augmentation and transitivity. Two sets of FDs F and H are said to be equivalent $F \equiv H \iff F^+ = H^+$.

Let F be a set of FDs from a database T, F is said to be *sound* if all FDs in F hold in T. In addition, F is said to be *complete* if all FDs that hold in T can be derived from F. Let $X \to Y$ be any FD in F, then F is said to be *non-redundant* if $F \backslash \{X \to Y\} \not\equiv F$, and *non-redundant w.r.t. augmentation* iff $(F \backslash \{X \to Y\})^* \neq F^*$

A set F is said to be *left-reduced* if for all $X \to Y \in F$ and $Z \subsetneq X$ we have that $(F \backslash \{X \to Y\}) \cup \{Z \to Y\} \not\equiv F$. Dually, it is said to be *right-reduced* if for all $X \to Y \in F$ and $Z \subsetneq Y$ we have that $(F \backslash \{X \to Y\}) \cup \{X \to Z\} \not\equiv F$. F is said to be *reduced* if it is simultaneously *left* and *right-reduced*.

Let F be a reduced set of FDs, then $G = \{X \to y \mid X \to Y \in F, y \in Y\}$ is the *splitting* of F and $G \equiv F$. Let F be a *reduced* set, its splitting is called a *canonical cover*. A *canonical cover* is a *left-reduced*, *non-redundant w.r.t. augmentation* set of FDs with a single element in their right hand side (a split set) [19]. A different definition presents the canonical cover in a *saturated* version requiring uniqueness in their left hand side [17] losing the single element condition in the right hand side. For the sake of compatibility with FCA implication covers we will favor the first definition. Notice that *canonical covers* can be redundant w.r.t. transitivity. For example the *canonical cover* of Table 1 contains $\{c \to b, c \to e, d \to e, bd \to c, be \to c\}$ where $bd \to c$ would be redundant w.r.t. transitivity as $bd \to bde \to c$.

Finally, a set F is said to be a *minimum cover* if it has as few FDs as any other equivalent set of FDs. For example, the *minimum cover* of Table 1 contains FDs $\{c \to be, d \to e, be \to c\}$. Notice that the minimum cover is not restricted to be reduced, so it is not presented with split sets. Secondly, the minimum cover contains exactly one fewer FD than the canonical cover, namely $bd \to c$.

2.2 Formal Concept Analysis, Implication Systems and FDs

For the sake of brevity we do not provide a description of the Formal Concept Analysis (FCA) framework. The notation used in this article follows [13] where $\mathcal{K} = (G, M, I)$ is a formal context of objects G, attributes M and incidence relation I, with formal concepts (A, B) where $A' = B$ and $B' = A$.

Implications are relations established between attribute sets from a formal context $\mathcal{K}$. Implications are analogous to FDs and they can be used to characterize them [2]. Because of this, we will notate an implication similarly to an FD. Implication systems (sets of implications) can also characterize FD covers [4].

An implication $X \to Y$ holds in $\mathcal{K}$ for $X, Y \subseteq M$ if $Y \subseteq X''$. Let T be a set of tuples and $\mathcal{U}$, a set of attributes in a table (such as the one in Table 1). We define the set $Pair(T) = \binom{T}{2}$ set of all subsets T with exactly two elements, and the incidence set I such that $((t_i, t_j), x) \in I \iff t_i[x] = t_j[x]$, $\forall x \in \mathcal{U}$, $\forall t_i, t_j \in T$. It can be shown that an FD $X \to Y$ holds in the database if and only if $X \to Y$ is an implication of the formal context $\mathcal{K} = (Pair(T), \mathcal{U}, I)$ [13].

$\mathcal{K}$ is called the *binary codification* of table T. For example, Table 3 contains the binary codification of Table 1. In Table 3 we can observe the implication $d \rightarrow e$ which can be verified as an FD in Table 1.

The previous statement entails that FDs and implications in $\mathcal{K}$ are in 1-1 correspondence. Moreover, the corresponding definition of a canonical cover of FDs is equivalent to that of a *canonical-direct unitary basis* of implications as shown in [4] where the equivalent *left-minimal basis* is described as containing *minimal functional dependencies*, i.e. those in a canonical cover.

Table 1: Example of a table T

	a	b	c	d	e
t_1	1	1	1	1	1
t_2	2	1	1	1	1
t_3	3	1	2	2	2
t_4	3	2	3	2	2
t_5	3	1	2	3	2
t_6	1	1	2	2	2
t_7	1	1	2	4	2

Table 2: Representation context

	a	b	c	d	e
$\delta(a)$	×				
$\delta(b)$		×			
$\delta(e)$					×
$\delta(a) \sqcap \delta(b)$	×	×	↙		↙
$\delta(a) \sqcap \delta(e)$	×				×
$\delta(d) \sqcap \delta(e)$				×	×
$\delta(a) \sqcap \delta(d) \sqcap \delta(e)$	×	↙	↙	×	×
$\delta(b) \sqcap \delta(c) \sqcap \delta(e)$		×	×		×
$\delta(a) \sqcap \delta(b) \sqcap \delta(c) \sqcap \delta(e)$	×	×	×	↙	×
$\delta(b) \sqcap \delta(c) \sqcap \delta(d) \sqcap \delta(e)$	↙	×	×	×	×
$\delta(a) \sqcap \delta(b) \sqcap \delta(c) \sqcap \delta(d) \sqcap \delta(e)$	×	×	×	×	×

Table 3: Binary codification of Table 1

	a	b	c	d	e
(t_1, t_2)		×	×	×	×
(t_1, t_3)		×			
(t_1, t_4)					
(t_1, t_5)		×			
(t_1, t_6)	×	×			
(t_1, t_7)	×	×			
(t_2, t_3)		×			
(t_2, t_4)					
(t_2, t_5)		×			
(t_2, t_6)		×			
(t_2, t_7)		×			
(t_3, t_4)	×			×	×
(t_3, t_5)	×	×	×		×
(t_3, t_6)		×	×	×	×
(t_3, t_7)		×	×		×
(t_4, t_5)	×				×
(t_4, t_6)				×	×
(t_4, t_7)					×
(t_5, t_6)		×	×		×
(t_5, t_7)		×	×		×
(t_6, t_7)	×	×	×		×

2.3 Partition Pattern Structures

A Partition Pattern Structure (PPS) is a type of pattern structure [14] that deals with, as the name suggests, object representations in the form of set partitions. PPS have shown to be useful for mining biclusters [7] and, more importantly, relations between partition pattern concepts have been used to characterize FDs of different kinds [3].

The formalization of a database T with attributes $\mathcal{U}$ as a PPS is as follows. A partition d of T is a set $\mathrm{d} \subseteq \wp(T)$ (powerset of T) of disjoint non-empty subsets of T such that for any two different elements $K_i, K_j \in \mathrm{d}$ we have that $K_i \cap K_j = \varnothing$ and $\bigcup_{K \in \mathrm{d}} K = T$. Let D be the set of all possible partitions of T, then any two partitions $\mathrm{d}_1, \mathrm{d}_2 \in \mathrm{D}$ can be ordered by a *coarser/finer* relation denoted $\mathrm{d}_1 \sqsubseteq \mathrm{d}_2$ (d_1 is finer than d_2) iff $(\forall\ K_i \in \mathrm{d}_1)(\exists\ K_j \in \mathrm{d}_2)$ such that $K_i \subseteq K_j$. The similarity operator is defined as $\mathrm{d}_1 \sqcap \mathrm{d}_2 = \{K_i \cap K_j \mid K_i \in \mathrm{d}_1, K_j \in \mathrm{d}_2\}$. From this, it follows that $(\mathrm{D}, \sqsubseteq)$ is a complete lattice with supremum and infimum defined respectively as $\top = \{\{T\}\}$ and $\bot = \{\{t\} \mid t \in T\}$.

The set of attributes $\mathcal{U}$ is mapped onto D through a function δ which for a given attribute $x \in \mathcal{U}$ yields a partition $\delta(x) \in \mathrm{D}$ representing the equivalence

relations over T for values of attribute x. With this, we can configure the PPS $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ with derivation operators $X^{\square} = \bigsqcap_{x \in X} \delta(x)$ denoting the equivalence relations implied by attributes in X, and $\mathrm{d}^{\square} = \{x \in \mathcal{U} \mid \delta(x) \sqsubseteq \mathrm{d}\}$ denoting all attributes with associated equivalence relations finer than d. (X, d) is a partition pattern concept when $X^{\square} = \mathrm{d}$ and $\mathrm{d}^{\square} = X$.

Theorem 1 (Proposition 2 in [3]). *Let $(Pair(T), \mathcal{U}, I)$ be the binary codification of a table within a database, and $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ its PPS representation:*

$$(W, X) \in (Pair(T), \mathcal{U}, I) \iff (X, \mathrm{d}) \in (\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$$

The proof of Theorem 1 can be found in [3]. Theorem 1 presents an important property of the PPS that states that $X \subseteq \mathcal{U}$ is an extent in $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ if and only if it is also an intent in $(Pair(T), \mathcal{U}, I)$ (the relation between the set of tuple pairs $W \subseteq Pair(T)$ and the partition $\mathrm{d} \in \mathrm{D}$ can be formalized as well but it is of no interest to our development). Theorem 1 is very important since it entails that the lattices derived from $(Pair(T), \mathcal{U}, I)$ and $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ are isomorphic. Consequently, implication $X \to Y$ in $(Pair(T), \mathcal{U}, I)$ can be found as a relation between extents in $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ (extent implication) such that $Y \subseteq X^{\square\square}$.

3 Covers and Pattern Structures

In this section we present two different kinds of covers for FDs: canonical covers (Section 3.1) and minimal covers (Section 3.2), as well as their characterization in terms of Pattern Structures. This section uses existing well-known results in FCA, which are reviewed here for the sake of readability.

Section 3.1 presents how a canonical cover for FDs can be computed using PPS, according to the results in [4, 22]. Section 3.2 introduces a novel characterization of the minimum cover of FDs by means of pseudo-extents of PPS [13]. The interest of these results is not limited to computing FD covers, but also for generalizations of FDs [3, 6].

3.1 Characterizing a Canonical Cover of FDs

The characterization of a canonical cover of FDs using FCA is straightforward. In a nutshell, a canonical cover of FDs is analogous to a *canonical direct unitary basis* of implications [4] as presented in Section 2.2. In this section we recall some of these ideas and show how they can be simply adapted to the framework of PPS.

Firstly, let $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ be a PPS, we define $\underline{\mathrm{D}} = \{\mathrm{d} \in \mathrm{D} \mid \mathrm{d}^{\square\square} = \mathrm{d}\}$ as the set of all closed partition patterns in D. The formal context $(\mathcal{U}, \underline{\mathrm{D}}, J)$ with $(\mathrm{d}, x) \in J \iff \mathrm{d} \sqsubseteq \delta(x)$ is called a representation context of $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ and their corresponding concept lattices are isomorphic [18]. For the sake of readability of the following definitions, we define the representation context as $(\underline{\mathrm{D}}, \mathcal{U}, J)$ instead of $(\mathcal{U}, \underline{\mathrm{D}}, J)$ (Table 2). By transitivity of equivalence, it is clear

that $(\underline{\mathrm{D}},\mathcal{U},J)$ is isomorphic to $(Pair(T),\mathcal{U},I)$ as defined in Section 2.3 and as such, implications in $(\underline{\mathrm{D}},\mathcal{U},J)$ correspond to FDs. For example, Table 2 (not considering elements $\swarrow$) contains the representation context $(\underline{\mathrm{D}},\mathcal{U},J)$ of the PPS derived from Table 1. Notice that objects are closed intersections of object representations, e.g. $\delta(d)$ does not appear since $\delta(d)^{\square\square} = \delta(d) \sqcap \delta(e)$. With this, the canonical direct basis of implications in $(\underline{\mathrm{D}},\mathcal{U},J)$ (and thus, canonical cover of FDs) is determined by the set of proper premises of elements in $\mathcal{U}$.

Theorem 2 (Corollary 1 in [22]). *$X \subseteq \mathcal{U}\backslash\{y\}$ is a premise of $y \in \mathcal{U}$ iff X is a hypergraph transversal of $\mathcal{H}_y^{\swarrow}$ defined as :*

$$\mathcal{H}_y^{\swarrow} = \{((\mathcal{U}\backslash\{y\}))\backslash \mathrm{d}' \mid \mathrm{d} \in \underline{\mathrm{D}}, \mathrm{d} \swarrow y\}$$

The set of all proper premises of y is the minimum hypergraph transversal $Tr(\mathcal{H}_y^{\swarrow})$.

A detailed description on the development of Theorem 2 can be found in [22]. Providing a formal definition of hypergraph transversals is out of the scope of this article, however we briefly mention that this formalization can also be made considering set collections (instead of hypergraphs) and minimum hitting sets (instead of minimum hypergraph transversals) [11]. This problem is also analogous to the vertex cover problem [12].

Theorem 2 provides a formal description for the proper premises of a given attribute $y \in \mathcal{U}$ that in turn yields the canonical cover of functional dependencies. However this approach requires the creation of the representation context which is a middle step in the overall calculation process. Actually, by analyzing the arrow relation between d and y we can observe that the representation context is not necessary. Consider that in $(\underline{\mathrm{D}},\mathcal{U},J)$, $\mathrm{d}' = \{x \in \mathcal{U} \mid (\mathrm{d},x) \in J(\iff \mathrm{d} \sqsubseteq \delta(x))\}$ and thus d' is equivalent to $\mathrm{d}^{\square}$ for any $\mathrm{d} \in \underline{\mathrm{D}}$. Moreover, in $(\mathcal{U},(\mathrm{D},\sqsubseteq),\delta)$, $\mathrm{d}^{\square} \subsetneq \mathrm{h}^{\square} \iff \mathrm{h} \subsetneq \mathrm{d}$ since $\mathrm{h} = \mathrm{h}^{\square\square}$ and $\mathrm{d} = \mathrm{d}^{\square\square}$. With this, we can rewrite the arrow definition as follows.

$$\begin{aligned}
\mathrm{d} \swarrow y &\iff (\mathrm{d},y) \notin J \text{ and if } \mathrm{d}' \subsetneq \mathrm{h}' \text{ then } (\mathrm{h},y) \in J \\
&\iff \mathrm{d} \not\sqsubseteq \delta(y) \text{ and if } \mathrm{d}^{\square} \subsetneq \mathrm{h}^{\square} \text{ then } \mathrm{h} \sqsubseteq \delta(y) \\
&\iff \mathrm{d} \not\sqsubseteq \delta(y) \text{ and if } \mathrm{h} \subsetneq \mathrm{d} \text{ then } \mathrm{h} \sqsubseteq \delta(y)
\end{aligned}$$

The last result shows that $\mathrm{d} \swarrow y$ in $(\underline{\mathrm{D}},\mathcal{U},J)$ can be defined directly over the PPS. Intuitively, this definition corresponds to $y \nearrow \mathrm{d}$ in $(\mathcal{U},\underline{\mathrm{D}},J)$ and thus, in $(\mathcal{U},(\mathrm{D},\sqsubseteq),\delta)$. With these elements we can finally propose a characterization for the canonical cover of functional dependencies in $(\mathcal{U},(\mathrm{D},\sqsubseteq),\delta)$ as follows.

Corollary 1. *Let $(\mathcal{U},(\mathrm{D},\sqsubseteq),\delta)$ be a partition pattern structure and $Tr(\mathcal{H})$ denote the hypergraph transversal of $\mathcal{H}$, then with*

$$\begin{aligned}
\mathcal{L}_{cc} &= \{X \to y \mid y \in \mathcal{U}, X \in Tr(\mathcal{H}_y^{\nearrow})\} \\
\mathcal{H}_y^{\nearrow} &= \{((\mathcal{U}\backslash\{y\}))\backslash \mathrm{d}' \mid \mathrm{d} \in \underline{\mathrm{D}}, y \nearrow \mathrm{d}\} \\
y \nearrow \mathrm{d} &\iff \mathrm{d} \not\sqsubseteq \delta(y) \textit{ and if } \mathrm{h} \subsetneq \mathrm{d} \textit{ then } \mathrm{h} \sqsubseteq \delta(y)
\end{aligned}$$

$\mathcal{L}_{cc}$ is a canonical cover of functional dependencies.

For the running example, let us calculate the proper premises of attribute c using the arrow relations in Table 3. We have $c \nearrow (\delta(a) \sqcap \delta(b))$ and $c \nearrow (\delta(a) \sqcap \delta(d) \sqcap \delta(e))$. With this, we have the hypergraph $\mathcal{H}_c^{\nearrow} = \{\{d, e\}, \{b\}\}$ for which the minimum transversal hypergraph is $Tr(\mathcal{H}_c^{\nearrow}) = \{\{b, d\}, \{b, e\}\}$. Correspondingly, we have the FDs $bd \rightarrow c$ and $be \rightarrow c$ which are included in the canonical cover.

3.2 Characterizing a Minimal Cover of FDs

We introduce a novel characterization of a minimal cover of FDs by means of pseudo-intents, and its generalization using pseudo-extents of a PPS.

The stem base of implications, or Duquenne-Guigues basis [15], is a *sound*, *complete* and *non-redundant* basis which also has minimum cardinality among the sets of implications for a given formal context. We show how this can be used to characterize a minimal cover of FDs in a rather simple manner. Prior to introducing the stem base, let us define pseudo-closed sets [13].

Definition 2. *(Pseudo-closed sets) Let $P \mapsto P''$ be a closure operator over a set M, then P is a pseudo-closed set if and only if:*

$$P \neq P'' \tag{1}$$

$$Q \subsetneq P \text{ and } Q \text{ is a pseudo-closed set} \implies Q'' \subseteq P \tag{2}$$

Given a formal context (G, M, I), pseudo-closed sets $A \subseteq G$ are called pseudo-extents, while pseudo-closed sets $B \subseteq M$ are called pseudo-intents. A stem base of implications, or Duquenne-Guigues basis, can be defined as follows:

Theorem 3 ([13]). *(Duquenne-Guigues Basis) The set of implications:*

$$\mathcal{L} = \{P \rightarrow P'' \mid P \text{ is a pseudo-intent}\} \tag{3}$$

is sound, complete and non-redundant.

Theorem 4 ([13]). *Every complete set of implications contains an implication $X \rightarrow Y$ with $X'' = P''$ for every pseudo-intent P of (G, M, I)*

Theorem 4 entails that the stem base of implications has minimum cardinality with respect to any equivalent set of implications of (G, M, I). With this and the previous observation that FDs are in 1-1 correspondence with implications in $(Pair(T), \mathcal{U}, I)$, we can derive the following corollary.

Corollary 2. *Let $(Pair(T), \mathcal{U}, I)$ be the binary codification of a table with tuples T and attributes $\mathcal{U}$, the set of FDs $\mathcal{L} = \{P \rightarrow P'' \mid P \text{ is a pseudo-intent}\}$ is a minimal cover.*

Corollary 2 provides a novel characterization of the minimal cover of FDs through pseudo-intents of a formal context. Given the existing relation between $(Pair(T), \mathcal{U}, I)$ and $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$, we can generalize the characterization of the minimal cover over the latter. Observe that in the PPS $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$, we maintain the notion of pseudo-extents for a pseudo-closed set $X \subseteq \mathcal{U}$ with $X \mapsto X^{\square\square}$.

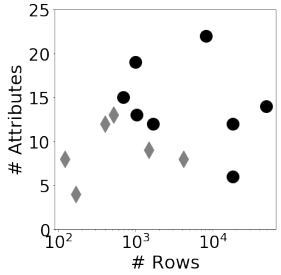

Fig. 1: Datasets and the number of rows and columns they contain.

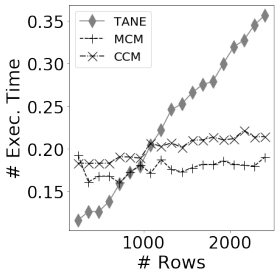

Fig. 2: Performance Comparison when increasing tuples: FCA vs TANE

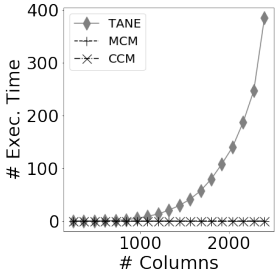

Fig. 3: Performance Comparison when increasing attributes: FCA vs TANE

Proposition 1. *Let* $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$ *be the PPS representation of a database, then the set of functional dependencies* $\mathcal{L}_{mc}$ *defined below is a minimal cover.*

$$\mathcal{L}_{mc} = \{X \rightarrow X^{\square\square} \mid X \textit{ is a pseudo-extent}\} \tag{4}$$

The proof of Proposition 1 follows from Theorem 1 and the fact that for a set $X \in \mathcal{U}$ the closure operator $X \mapsto X''$ is exactly equivalent to $X \mapsto X^{\square\square}$ and consequently, the set of pseudo-intents in $(Pair(T), \mathcal{U}, I)$ is the same as the set of pseudo-extents in $(\mathcal{U}, (\mathrm{D}, \sqsubseteq), \delta)$. Thus, because of Corollary 2, Proposition 1 holds.

Table 4: Dataset details, Execution Times in Seconds, and Number of Mined Rules for CCM (Canonical Cover Miner), MCM (Minimal Cover Miner) and TANE. CC: Canonical Cover, MC: Minimal Cover. Datasets in boldface represent those in which FCA performed better than TANE.

			CCM		MCM		TANE	
Dataset	# Tuples	# Attributes	# CC Deps	Time [S]	# MC Deps	Time [S]	# CC Deps	Time [S]
Mushroom	8124	22	3605	23887	1509	12684	-	-
Adult	48842	14	78	90.41	42	71.55	78	123.13
Credit	690	15	1099	3.07	253	1.82	1099	2.54
PGLW	17995	6	5	0.67	2	0.35	5	0.48
PGLW (2xA)	17995	12	38	1.81	15	1.18	38	7.45
Forest Fires	516	13	442	0.46	138	0.31	442	0.49
Forest Fires (2xT)	1032	13	442	1.27	138	0.78	442	2.34
ncvoter	1000	19	775	2.47	193	1.63	775	2.07
Diagnostics	120	8	37	0.08	17	0.06	37	0.06
Abalone	4177	8	137	0.41	40	0.29	137	0.32
CMC	1473	9	1	0.56	1	0.49	1	0.52
Hughes	401	12	3	0.12	3	0.17	3	0.06
Servo	167	4	1	0.05	1	0.03	1	0.02
Caulkins	1685	12	227	0.66	67	0.53	227	0.95

4 Experimental Evaluation

In this section we present a brief experimental comparison of both introduced approaches versus TANE [16], a state-of-the-art FD miner. TANE is a highly optimized *apriori*-based algorithm that generates a canonical cover of FDs. The goal of this evaluation is to study the comparative benefits of using FCA versus a traditional approach such as TANE. TANE was re-implemented for our experiments[5]. For the sake of repeatibility, the code used to run the experiments was made available at GitHub. Both the minimal cover miner (MCM) [6], and the canonical cover miner (CCM)[7] were implemented using Python's pipy FCA library `fca`[8].

Experiments were performed over 12 datasets extracted from the UCI Machine Learning repository[9], the JASA Data Archive[10] and Metanome's repeatability Web page[11]. Details on the number of rows and columns for each dataset are provided in the first two columns of Table 4. In addition to these datasets, we created synthetic versions by multiplying the rows or the columns of a given dataset. Experiments were performed on a virtual machine with 4 cores running at 2.7 Ghz and equipped with 32 GB of RAM memory.

4.1 Results & Discussion

Table 4 presents the main results of applying our approach and TANE on each dataset to mine the Minimal Cover and the Canonical Cover, respectively. The table contains the execution times for each approach and the number of dependencies mined. Datasets in boldface represent those for which CCM or MCM performed substantially better than TANE. For the Mushroom dataset, TANE was not able to obtain results before running out of memory, thereby no information is provided in the table. Table 4 also reports in two synthetic datasets, namely PGLW (2xA) which contains two horizontal copies of the PGLW dataset resulting in twice as many attributes. Forest Fires (2xT) contains two vertical copies of Forest Fires resulting in twice as many tuples. All Canonical Covers mined by TANE have been reduced to a Minimal Cover to verify the consistency of our approach.

Out of the 12 datasets, CCM and MCM performed better in the **largest** (both in rows and columns). This is better illustrated in Figure 1 where datasets are represented as points in a *number of rows-number of columns* space. Circles represent datasets for which CCM or MCM performed better while diamonds, where TANE did. Notice that the X axis is provided in logarithmic scale. The figure shows that most of the datasets where TANE performs better are in

[5] `https://github.com/codocedo/tane/tree/cla18`
[6] `https://github.com/codocedo/fd_miner/tree/cla18`
[7] `https://github.com/codocedo/uis_miner/tree/cla18`
[8] `https://pypi.org/project/fca/`
[9] `https://archive.ics.uci.edu/ml/index.php`
[10] `http://lib.stat.cmu.edu/jasadata/`
[11] `https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html`

the lower-left region of the figure, representing small datasets. FCA-based approaches perform better in datasets in all other regions, including the upper-right which contains datasets with many rows and columns.

Synthetic datasets in Table 4 show evidence that FCA scales better when duplicating the dataset. When duplicating attributes the difference is particularly dramatic since TANE is over 13 times slower while our approach, only 3. To study this further, we created two sets of synthetic datasets. The first set (vertical set) was created by incrementally multiplying vertically the *Diagnostics* datasets (with 8 attributes and 120 tuples) generating 19 versions of 240, 360, 480 tuples, up to a dataset containing 2400 tuples. The second set (horizontal set) was created using the same idea but in a horizontal manner generating 19 versions of 16, 24, 32, up to 160 attributes. Since most of the datasets of the second set were too big for TANE, they were trunked to 40 tuples.

Figure 2 depicts the increasing time for CCM, MCM and TANE on the vertical set, i.e. when increasing the number of tuples. We can observe that all three approaches scale linearly w.r.t. the number of tuples, even when CCM and MCM seem to have a more stable behavior. Vertical multiplication of datasets yield the same number of FDs than the original set, since the relation between attributes remains unchanged. Thus, we can assume that other algorithms based on TANE could achieve a similar performance to CCM or MCM provided some optimizations.

On the other hand, this do not seem to be the case for the horizontal set. Figure 3 shows that CCM and MCM remain very stable when varying the number of attributes, while TANE's execution time grows exponentially. Indeed, this great difference in performance is due to the way in which we use FCA to find FDs which differs from TANE's strategy. Using FCA we calculate closures which quickly group attribute copies avoiding unnecessary intersections. Instead, TANE computes each attribute combination rendering the exponential growth in the computation time. We stress that this is not simply an extreme case from which our approach takes advantage, but actually a very good illustration of the benefits of using a closure operator to navigate the space of FDs. Closures enable CCM and MCM to avoid unnecessary computations not only when we have redundant attributes, but also whenever possible in the lattice of the powerset of attributes. Finally, we do not discuss on the differences between CCM and MCM strategies as these are detailed in [22].

5 Conclusions

We have presented a new characterization of a minimal cover of functional dependencies (FDs) by means of the stem base (or Duquenne-Guigues basis) of a partition pattern structure. We have presented the mechanisms through which this characterization can be exploited to efficiently mine the minimal cover. Furthermore, we have described the algorithms that implement these mechanisms and show empirical evidence that our characterization performs better than a

state-of-the-art FD miner, namely TANE, in larger databases containing many rows and columns.

Acknowledgments. This research work has been supported by the SGR2014-890 (MACDA) project of the Generalitat de Catalunya, and MINECO project APCOM (TIN2014-57226-P).

References

1. Alam, M., Buzmakov, A., Codocedo, V., Napoli, A.: Mining definitions from RDF annotations using formal concept analysis. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 823–829. AAAI Press (2015)
2. Baixeries, J., Kaytoue, M., Napoli, A.: Computing Functional Dependencies with Pattern Structures. In: Concept Lattices and their Applications. pp. 175–186 (2012)
3. Baixeries, J., Kaytoue, M., Napoli, A.: Characterizing Functional Dependencies in Formal Concept Analysis with Pattern Structures. Annals of Mathematics and Artificial Intelligence 72(1–2), 129–149 (Oct 2014)
4. Bertet, K., Monjardet, B.: The multiple facets of the canonical direct unit implicational basis. Theoretical Computer Science 411(22-24), 2155–2166 (2010)
5. Caruccio, L., Deufemia, V., Polese, G.: Relaxed Functional Dependencies – A Survey of Approaches. IEEE Transactions on Knowledge and Data Engineering 28(1), 147–165 (2016)
6. Codocedo, V., Baixeries, J., Kaytoue, M., Napoli, A.: Characterization of Order-like Dependencies with Formal Concept Analysis. In: Concept Lattices and Their Applications. vol. 1624, pp. 123–134 (2016)
7. Codocedo, V., Napoli, A.: Lattice-based biclustering using Partition Pattern Structures. In: Proceedings of ECAI 2014. Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 213–218. IOS Press (2014)
8. Fan, W.: Dependencies revisited for improving data quality. In: Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada. pp. 159–170 (2008)
9. Fan, W.: Data quality: From theory to practice. SIGMOD Record 44(3), 7–18 (2015)
10. Fan, W., Geerts, F.: Foundations of Data Quality Management. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2012)
11. Gainer-Dewar, A., Vera-Licona, P.: The minimal hitting set generation problem: Algorithms and computation. SIAM Journal on Discrete Mathematics 31(1), 63–100 (2017)
12. Ganter, B., Obiedkov, S.: Conceptual Exploration. Springer, Berlin (2016)
13. Ganter, B., Wille, R.: Formal Concept Analysis. Springer, Berlin (1999)
14. Ganter, B., Kuznetsov, S.O.: Pattern Structures and their projections. In: Delugach, H.S., Stumme, G. (eds.) Conceptual Structures: Broadening the Base. LNCS, vol. 2120, pp. 129–142. Springer (2001)
15. Guigues, J.L., Duquenne, V.: Familles minimales d'implications informatives résultant d'un tableau de données binaires. Mathématiques et Sciences Humaines 95, 5–18 (1986)

16. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. Computer Journal 42(2), 100–111 (1999)
17. Korth, H.F., Silberschatz, A.: Database System Concepts. McGraw-Hill, Inc., New York, NY, USA (1986)
18. Kuznetsov, S.O.: Galois Connections in Data Analysis: Contributions from the Soviet Era and Modern Russian Research. In: Formal Concept Analysis, Foundations and Applications. pp. 196–225. Lecture Notes in Computer Science 3626, Springer (2005)
19. Maier, D.: Theory of Relational Databases. Computer Science Press (1983)
20. Mannila, H., Räihä, K.J.: The Design of Relational Databases. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1992)
21. Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J.P., Schönberg, M., Zwiener, J., Naumann, F.: Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. Proceedings of VLDB Endowment 8(10), 1082–1093 (2015)
22. Ryssel, U., Distel, F., Borchmann, D.: Fast algorithms for implication bases and attribute exploration using proper premises. Annals of Mathematics and Artificial Intelligence 70(1), 25–53 (2014)
23. Ullman, J.: Principles of Database Systems and Knowledge-Based Systems, volumes 1–2. Computer Science Press, Rockville (MD), USA (1989)
24. Yu, Y., Heflin, J.: Extending functional dependency to detect abnormal data in RDF graphs. In: The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I. pp. 794–809 (2011)
25. Yu, Y., Li, Y., Heflin, J.: Detecting abnormal semantic web data using semantic dependency. In: Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC 2011), Palo Alto, CA, USA, September 18-21, 2011. pp. 154–157. IEEE Computer Society (2011)