Characterizing Approximate-Matching Dependencies in Formal Concept Analysis with Pattern Structures

Jaume Baixeries[a] , Victor Codocedo[c], Mehdi Kaytoue[c,d] and Amedeo Napoli[b]

[a]*Departament de Ciències de la Computació. Universitat Politècnica de Catalunya. 08032 Barcelona. Catalonia;*
[b]*LORIA (CNRS – Inria Nancy Grand-Est – Université de Lorraine) B.P. 239, 54506 Vandœuvre-lès-Nancy, France;*
[c]*Université de Lyon. CNRS, INSA-Lyon, LIRIS. UMR5205, F-69621, France*
[d]*Infologic, 99 Avenue de Lyon, F-26500, Bourg-lès-Valence, France*

Abstract

Functional dependencies (FDs) provide valuable knowledge on the relations between attributes of a data table. A functional dependency holds when the values of an attribute can be determined by another. It has been shown that FDs can be expressed in terms of partitions of tuples that are in agreement w.r.t. the values taken by some subsets of attributes. To extend the use of FDs, several generalizations have been proposed. In this work, we study approximate-matching dependencies that generalize FDs by relaxing the constraints on the attributes, i.e. agreement is based on a similarity relation rather than on equality. Such dependencies are attracting attention in the database field since they allow uncrisping the basic notion of FDs extending its application to many different fields, such as data quality, data mining, behavior analysis, data cleaning or data partition, among others. We show that these dependencies can be formalized in the framework of Formal Concept Analysis (FCA) using a previous formalization introduced for standard FDs. Our new results state that, starting from the conceptual structure of a pattern structure, and generalizing the notion of relation between tuples, approximate-matching dependencies can be characterized as implications in a pattern concept lattice. We finally show how to use basic FCA algorithms to construct a pattern concept lattice that entails these dependencies after a slight and tractable binarization of the original data.

Keywords: functional dependencies, similarity, tolerance relation, formal concept analysis, pattern structures, attribute implications.

1. Introduction

In the relational database model, functional dependencies (FDs) are among the most popular types of dependencies since they indicate a functional relation between sets of attributes [1, 2, 3]: the values of a set of attributes are determined by the values of another set of attributes. Such FDs can be used to check the consistency and the quality of a database [4], but also to guide the database design [5].

However, the definition of FDs is too strict for several useful tasks, for instance when dealing with data imprecision i.e. errors and uncertainty in real-world data. To overcome this problem, different generalizations of FDs have been defined. These generalizations can be classified according to the criteria by which they relax the equality condition of FDs [6]. According to this classification, two main strategies are presented: "extent relaxation" and "attribute relaxation" (in agreement with the terminology introduced in [6]).

Characterizing and computing FDs are two tasks strongly related to lattice theory. For example, lattice characterizations of a set of FDs are studied in [7, 8, 9, 10]. Following the same line, a characterization of FDs within Formal Concept Analysis (FCA) is proposed in [11]. In the latter case, FDs are shown to be in one-to-one correspondence with the set of implications of a formal context (binary table) generated from a database. However, such a formal context has a quadratic number of objects w.r.t. the tuples of the original database. To avoid this, [12] and [13] show how to use pattern structures, introduced by [14] as an extension of FCA. Moreover, in [15] it is shown how this framework can be extended to Similarity Dependencies, another generalization of FDs.

Besides FCA and implications, there are many similarities between association rules in data mining and FDs. This is discussed further in the present paper and as well in [16]. In the latter, a unifying framework in which any "well-formed" semantics for rules may be integrated is introduced. Similarly, this is also what we try to define in this paper for generalizations of FDs in the framework of FCA and Pattern Structures.

This paper presents an extended and updated version of [15] and its main objective is to give a characterization of FDs relaxing the attribute comparison within FCA thanks to the formalism of Pattern Structures. While our previous work considered similarity dependencies, we extend the characterization to the family of approximate-matching dependencies using pattern structures and tolerance relations [17, 18]. Furthermore, we show that the classical FCA algorithms can be –almost directly– applied to compute approximate-matching dependencies.

This paper is organized as follows. In Section 2 we introduce our notations and the definition of FDs. We present other kinds of generalization of FDs in Section 3. In Section 4, we introduce tolerance relations and we show how the dependencies that are enumerated in Section 3 are based on tolerance relations. In Section 5 we propose a generic characterization and computation of approximate-matching dependencies in terms of Pattern Structures. In Section 6 we present a set of experiments to test the feasibility and scalability of extracting approximate-matching dependencies with pattern structures.

2. Notation and Functional Dependencies

We deal with datasets which are sets of tuples. Let $\mathcal{U}$ be a set of attributes and 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 $t : \mathcal{U} \mapsto Dom$ and a table T is a set of tuples $T \subseteq Dom^{|\mathcal{U}|}$. Sometimes the set notation is omitted and we write ab instead of $\{a, b\}$.

Given a tuple $t \in T$ and $X = \{x_1, x_2, \ldots, x_n\} \subseteq \mathcal{U}$, we have:

$$t[X] = \langle t(x_1), t(x_2), \ldots, t(x_n) \rangle$$

$t[X]$ is called the *projection* of X onto t. In Example 1, we have $t_2[\{a, c\}] = \langle t_2(a), t_2(c) \rangle = \langle 6, 6 \rangle$. The definition can also be extended to a set of tuples. Given a set of tuples $S \subseteq T$ and $X \subseteq \mathcal{U}$, we have:

$$S[X] = \{t[X] \mid t \in S\}$$

Example 1. *Table with tuples $T = \{t_1, t_2, t_3, t_4\}$ and attributes $\mathcal{U} = \{a, b, c, d\}$.*

id	*a*	*b*	*c*	*d*
t_1	*3*	*5*	*6*	*3*
t_2	*6*	*5*	*6*	*5*
t_3	*3*	*10*	*6*	*3*
t_4	*6*	*5*	*9*	*5*

We now formally introduce functional dependencies [3].

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

$$\forall t, t' \in T : t[X] = t'[X] \Rightarrow t[Y] = t'[Y]$$

For example, the functional dependencies $a \to d$ and $d \to a$ hold in the table of Example 1, whereas the functional dependency $a \to c$ does not hold since $t_2(a) = t_4(a)$ but $t_2(c) \neq t_4(c)$.

3. Generalizations of Functional Dependencies

Functional dependencies tell us which attributes are determined by other attributes. As such, FDs are mainly used in databases to determine which attributes are the *keys* of a dataset, i.e. the minimal sets of attributes (if any) determining all other attributes. This information is necessary for maintaining the consistency of the whole database. Moreover, this information can also be useful in data analysis or in data classification, because of the the semantics attached to the "determined by" relationship.

However, in practical applications we usually have datasets that contain imprecise or uncertain information. Here, we do not mean *false* information, but information that may contain errors. For example, let us consider a dataset containing information about the name and social security number (SSN) of citizens. Although SSN is supposed to be unique for each individual, it appears that sometimes SSN is shared by more than one individual. In such a case, the FD `‘‘social security number` $\rightarrow$ `name, surname’’` would not hold, although we know that SSN *does* determine an individual. This is a case where FDs reveal inconsistencies or errors in a database. Actually, here we rely on prior domain knowledge on the dataset, i.e. SSN is attached to a unique individual. Without this previous assumption, the fact that a FD such as `‘‘social security number` $\rightarrow$ `name, surname’’` does not hold, would not give us any relevant information.

In some other cases, FDs may fail to capture interesting relations among attributes. For example, let us consider a dataset containing information about which brand of beer and TV sitcom are preferred by customers, involving attributes such as `customer id, beer brand, tv sitcom`. Now, assume that people preferring a specific brand of beer, say `beer X`, also prefer `sitcom Y` in an overwhelming 95% of cases. This means that there are 5% of the cases in which a customer preferring `beer X` will prefer `sitcom Z` for example. These few cases prevent the FD `beer brand` $\rightarrow$ `sitcom` from holding, even if this dependency holds in a very large majority of the cases in the dataset.

To overcome the limitations of FDs, several generalizations have been introduced. These generalizations can be divided into two main groups, depending on the strategy to relax the semantics of FDs [6]. These two categories are (1) “extent-relaxing dependencies” and (2) “attribute-relaxing dependencies”. Extent-relaxing dependencies relax the conditions on their extents and can be likened to *association rules* in data mining [19]. Attribute-relaxing dependencies relax the conditions on the equality condition of attributes. Here we are more interested in attribute-relaxing dependencies.

More precisely, relaxation is applied to the equality condition $t[X] = t'[X]$ and $t[Y] = t'[Y]$ in the definition of FDs, which is replaced with a less constrained condition. For example, we may replace the condition $t[X] = t'[X]$ with a condition where both $t[X]$ and $t'[X]$ need to be *sufficiently close according to a set of attributes* X. Such a condition still induces a relation among the set of tuples as for FDs.

Then two categories of attribute-relaxing dependencies can be distinguished. Firstly “approximate-matching dependencies” are such that the attribute comparison is computed by an approximation function, e.g. a distance function among the values appearing in the dataset. Secondly, “order-like dependencies” are such that two values must be ordered instead of being *somehow* similar or close.

Hereafter we propose an extensive study of approximate-matching dependencies.

4. Characterization of Approximate-Matching Dependencies

Below we propose a minimal characterization of approximate-matching dependencies. This characterization will be sufficient for representing attribute-relaxing dependencies within the formalism of Pattern Structures.

4.1. Tolerance Relations

Firstly, we define a *tolerance* relation in a set T and then the associated *blocks of tolerance*. Tolerance relations are often used in the formalization of similarity [17, 18].

Definition 2. *A tolerance relation $\theta \subseteq T \times T$ on a set T is a reflexive (i.e. $\forall t \in T : t\theta t$) and symmetric (i.e. $\forall t_i, t_j \in T : t_i\theta t_j \iff t_j\theta t_i$) relation.*

Example 2. *An often used tolerance relation is the similarity that can be defined within a set of integer values. Given two integer values v_1, v_2 and a user-defined threshold ϵ: $v_1\theta v_2 \iff |v_1 - v_2| \leq \epsilon$.*

Practically, let $t(Month)$ denotes the value of attribute $Month$ for tuple t. The function $\Delta_{Month}(m_1, m_2)$ defines the tolerance relation θ_{Month} such as:

$$\Delta_{Month}(m_1, m_2) = min(|m_1 - m_2|, min(m_1, m_2) + 12 - max(m_1, m_2))$$

$$t_i\theta_{Month}t_j \iff \Delta_{Month}(t_i(Month), t_j(Month)) \leq 4$$

Then θ_{Month} is the tolerance relation that considers two tuples similar if they have values within 4 months of distance.

In [15] we used *tolerance relations* to compare attribute values. In the following, we continue in the same way and we introduce blocks of tolerance and operations on these blocks that will allow us to make precise the notion of "approximate-matching dependency".

4.2. Blocks of Tolerance

A tolerance relation induces *blocks of tolerance* as follows.

Definition 3. *Given a set T, a subset $K \subseteq T$ and a tolerance relation $\theta \subseteq T \times T$. K is a* block of tolerance *of θ if:*

1. $\forall t_i, t_j \in K : t_i\theta t_j$ *(pairwise correspondence)*
2. $\forall t_i \notin K, \exists t_j \in K : \neg(t_i\theta t_j)$ *(maximality)*

Given a set of tuples T and a set of attributes $\mathcal{U}$, for each attribute $x \in \mathcal{U}$, we define a tolerance relation θ_x on the values of x. The set of tolerance blocks induced by θ_x is denoted by T/θ_x.

Example 3. *For example, when $T = \{1, 2, 3, 4, 5\}$, θ is defined as above, i.e. $v_1\theta v_2 \iff |v_1 - v_2| \leq \epsilon$, and $\epsilon = 2$, then $T/\theta = \{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}\}$. T/θ is not a partition, as θ is not transitive.*

All of the tuples in a tolerance block $K \in T/\theta_x$ are pairwise *similar* according to their values w.r.t. x. Thus we have:

Property 1. $\forall K_i, K_j \in T/\theta : K_i \nsubseteq K_j$ *and* $K_j \nsubseteq K_i$ for all $i \neq j$

We can define a partial ordering on the set of all possible tolerance relations in a set T as follows:

Definition 4. *Let θ_1 and θ_2 two tolerance relations in the set T. We say that $\theta_1 \leq \theta_2$ if and only if $\forall K_i \in T/\theta_1 : \exists K_j \in T/\theta_2 : K_i \subseteq K_j$*

This relation is a partial ordering and induces a lattice classifying tolerance relations, or, equivalently, blocks of tolerance. Given two tolerance relations, θ_1 and θ_2, the meet and the join operations in this lattice are:

Definition 5. *Let θ_1 and θ_2 two tolerance relations in the set T.*
$\theta_1 \wedge \theta_2 = \theta_1 \cap \theta_2 = \max_T(\{K_i \cap K_j \mid K_i \in T/\theta_1, K_j \in T/\theta_2\})$
$\theta_1 \vee \theta_2 = \theta_1 \cup \theta_2 = \max_T(T/\theta_1 \cup T/\theta_2)$
where $max_T(.)$ returns the set of maximal subsets w.r.t. inclusion.

Example 4. *Based on Example 1, let us define the tolerance relation θ_m w.r.t. an attribute $m \in \{a, b, c, d\}$ as follows: $t_i \theta_m t_j \iff |t_i(m) - t_j(m)| \leq \epsilon$. Then, assuming that $\epsilon = 1$, we have:*

$$T/\theta_a = \{\{t_1, t_3\}, \{t_2, t_4\}\} \quad T/\theta_b = \{\{t_1, t_2, t_4\}, \{t_3\}\}$$
$$T/\theta_c = \{\{t_1, t_2, t_3\}, \{t_4\}\} \quad T/\theta_d = \{\{t_1, t_3\}, \{t_2, t_4\}\}$$

We can check the following meet and join operations:

$$\theta_a \wedge \theta_b = \{\{t_1\}, \{t_2, t_4\}, \{t_3\}\} \quad \theta_a \vee \theta_b = \{\{t_1, t_2, t_3, t_4\}\}$$
$$\theta_a \wedge \theta_c = \{\{t_1, t_3\}, \{t_2\}, \{t_4\}\} \quad \theta_a \vee \theta_c = \{\{t_1, t_2, t_3, t_4\}\}$$
$$\theta_b \wedge \theta_c = \{\{t_1, t_2\}, \{t_3\}, \{t_4\}\} \quad \theta_b \vee \theta_c = \{\{t_1, t_2, t_3, t_4\}\}$$

We can also extend the definition of a similarity relation on a single attribute to a similarity relation on sets of attributes. Given $X \subseteq \mathcal{U}$, the similarity relation θ_X is defined as follows:

$$(t_i, t_j) \in \theta_X \iff \forall x \in X : (t_i, t_j) \in \theta_x$$

Two tuples are similar w.r.t. a set of attributes X if and only if they are similar w.r.t. *each* attribute in X.

4.3. The definition of Approximate-Matching Dependencies

Definition 6. *Let $X, Y \subseteq \mathcal{U}$: $X \to Y$ be an approximate-matching dependency if and only if: $\forall t_i, t_j \in T : t_i \theta_X t_j \to t_i \theta_Y t_j$*

While an FD $X \to Y$ is based on equality of values, an approximate-matching dependency $X \to Y$ holds if and only if, each pair of tuples having *related* values w.r.t. attributes in X has *related values* w.r.t. attributes in Y.

Example 5. *Recalling again Example 1, we define the tolerance relation:*

$$t_i \theta_m t_j \iff |t_i(m) - t_j(m)| \leq 2$$

Then the following approximate-matching dependencies hold:

- $a \to d, ab \to d, abc \to d, ac \to d, b \to d, bc \to d, c \to d.$
- *It is interesting to notice that $b \to d$ is an approximate-matching dependency but not a functional dependency, as $t_1(b) = t_2(b)$ and $t_1(d) \neq t_2(d)$.*
- *Because of the same pair of tuples, the approximate-matching dependency $bcd \to a$ does not hold, as we have $t_1 \theta_{bcd} t_2$ but we do not have $t_1 \theta_a t_2$, since $|t_1(a) - t_2(a)| \not\leq 2$.*
- *By contrast, the functional dependency $bcd \to a$ holds because there is no pair of tuples t_i, t_j such that $t_i(bcd) = t_j(bcd)$.*

5. Approximate-Matching Dependencies in FCA

5.1. A Brief Introduction to Pattern Structures in FCA

A "Pattern Structure" allows a direct application of FCA over non-binary and complex data [14]. Formally, let G be a set of objects, let $(D, \sqcap)$ be a meet-semi-lattice of potential object descriptions and let $\delta : G \to D$ be a mapping associating each object with its description. Then $(G, (D, \sqcap), \delta)$ is a pattern structure. Elements of D are patterns and are ordered thanks to a subsumption relation $\sqsubseteq$, i.e. $\forall c, d \in D$, $c \sqsubseteq d \iff c \sqcap d = c$.

A pattern structure $(G, (D, \sqcap), \delta)$ is based on two derivation operators denoted as $(\cdot)^\square$. For $A \subseteq G$ and $d \in D$:

$$A^\square = \bigsqcap_{g \in A} \delta(g) \qquad\qquad d^\square = \{g \in G | d \sqsubseteq \delta(g)\}.$$

These operators form a Galois connection between $(\wp(G), \cap)$ and $(D, \sqcap)$, where $\wp(G)$ is the powerset of G. Pattern concepts of $(G, (D, \sqcap), \delta)$ are pairs of the form (A, d), $A \subseteq G$, $d \in D$, such that $A^\square = d$ and $A = d^\square$. For a pattern concept (A, d), d is a pattern intent and is the common description of all objects in A, the pattern extent.

When partially ordered by $(A_1, d_1) \leq (A_2, d_2) \Leftrightarrow A_1 \subseteq A_2 \ \ (\Leftrightarrow d_2 \sqsubseteq d_1)$, the set of all concepts forms a complete lattice called pattern concept lattice.

5.2. Characterization of Dependencies within Pattern Structures

Thanks to the formalism of pattern structures, approximate-matching dependencies can be characterized and computed in an elegant manner. A dataset can be represented as a pattern structure $(\mathcal{U}, (D, \sqcap), \delta)$ where $\mathcal{U}$ is the set of original attributes, and $(D, \sqcap)$ is the set of sets of tolerance blocks over the set of tuples T provided with the meet operation introduced in Definition 5. The description of an attribute $x \in \mathcal{U}$ is given by $\delta(x) = T/\theta_x$ which is given by the set of tolerance blocks.

Example 6. *An example of concept formation in the pattern structure is given as follows. Consider the table in Example 1. Starting from the set* $\{a,c\} \subseteq \mathcal{U}$ *and assuming that* $t_i\theta_x t_j \iff |t_i(x) - t_j(x)| \leq 2$ *for all attributes:*

$$\begin{aligned}
\{a,c\}^\square = \delta(a) \sqcap \delta(c) &= \{\{t_1,t_3\},\{t_2,t_4\}\} \sqcap \{\{t_1,t_2,t_3\},\{t_4\}\} \\
&= \{\{t_1,t_3\},\{t_2\},\{t_4\}\} \\
\{\{t_1,t_3\},\{t_2\},\{t_4\}\}^\square &= \{x \in \mathcal{U} | \{\{t_1,t_3\},\{t_2\},\{t_4\}\} \sqsubseteq \delta(x)\} = \{a,c\}
\end{aligned}$$

Hence, $(\{a,c\},\{\{t_1,t_3\},\{t_2\},\{t_4\}\})$ *is a pattern concept.*

Then the pattern concept lattice allows us to characterize all approximate-matching dependencies holding in $\mathcal{U}$:

Proposition 1. *An approximate-matching dependency* $X \to Y$ *holds in a table* T *if and only if:* $\{X\}^\square = \{X,Y\}^\square$ *in the pattern structure* $(\mathcal{U},(D,\sqcap),\delta)$.

Proof. First of all, it should be noticed that $(t,t') \in \{X\}^\square$ if and only if $t(X)\theta_X t'(X)$, or, equivalently, $\forall x \in X : t(x)\theta_x t'(x)$.

Moreover, we also have that $\{X,Y\}^\square \subseteq \{X\}^\square$ and that $\{X,Y\}^\square \subseteq \{Y\}^\square$ because of the anti-monotonicity of $(.^\square)$.

We first prove that if $X \to Y$ holds in T, then, $\{X\}^\square = \{X,Y\}^\square$, and in particular that $\{X\}^\square \subseteq \{X,Y\}^\square$. We take an arbitrary pair $(t,t') \in \{X\}^\square$, i.e. $t(X)\theta_X t'(X)$. Since $X \to Y$ holds, then $t(X)\theta_X t'(X) \longrightarrow t(Y)\theta_Y t'(Y)$, and thus $(t,t') \in \{Y\}^\square$. This finally implies that $(t,t') \in \{X,Y\}^\square$.

Conversely, suppose now that $\{X\}^\square = \{X,Y\}^\square$. Then, when $t,t' \in T$ is such that $(t,t') \in \{X\}^\square$ (i.e. $t(X)\theta_X t'(X)$), it comes that $(t,t') \in \{X,Y\}^\square$. As $\{X,Y\}^\square \subseteq \{Y\}^\square$, we can infer that $(t,t') \in \{Y\}^\square$ and thus, $t(Y)\theta_Y t'(Y)$. Then we can conclude that $X \to Y$ holds in T. ∎

This proposition is structurally the same as the one used in [13] to prove the characterization of functional dependencies with pattern structures. In this case, we have changed and *relaxed* the equality condition by the tolerance relation θ.

5.3. Characterization of Dependencies with Binarization in FCA

In [12] it is shown how the *binarization* of a tabular dataset can be performed. Binarization is also defined in [2] and [5] as *agree sets*, which are pairs of tuples that agree in all the values of a given set of attributes. The binarization procedure can be formalized as follows.

For a given table with tuples T and a set of attributes $\mathcal{U}$, we define a binary context $\mathbb{K} = (Pair(T),\mathcal{U},I)$, where $Pair(T) = \{t_i,t_j \mid i < j\}$ and $((t_i,t_j),x) \in I$ (for all $x \in \mathcal{U}$) if and only if $t_i(x) = t_j(x)$. We generalize this notion of binarization to include a tolerance relation θ. To accomplish this it suffices to define the formal context $\mathbb{K}_\theta = (Pair(T),\mathcal{U},I_\theta)$ where $((t_i,t_j),x) \in I_\theta \iff t_i\theta_x t_j$.

Figure 1 shows an example of a dataset T (left) and its associate classical binarization $\mathbb{K} = (Pair(T),\mathcal{U},I)$ (middle). The formal context $\mathbb{K}_\theta =$

id	a	b	c	d
t_1	1	2	3	1
t_2	1	2	1	4
t_3	1	1	3	4
t_4	2	2	3	4

id	a	b	c	d
(t_1, t_2)	x	x		
(t_1, t_3)	x		x	
(t_1, t_4)		x	x	
(t_2, t_3)	x			x
(t_2, t_4)		x		x
(t_3, t_4)			x	x

id	a	b	c	d
(t_1, t_2)	x	x		
(t_1, t_3)	x	x	x	
(t_1, t_4)	x	x	x	
(t_2, t_3)	x	x		x
(t_2, t_4)	x	x		x
(t_3, t_4)	x	x	x	x

Figure 1: A data table T (left) with associated binarized formal context $(Pair(T), \mathcal{U}, I)$ (middle), and the generalization of the binarized formal context $(Pair(T), \mathcal{U}, I_\theta)$ taking $t_i \theta_x t_j \iff | t_i(x) - t_j(x) | \leq 1$.

$(Pair(T), \mathcal{U}, I_\theta)$ in Figure 1 (right) is obtained when defining $t_i \theta_x t_j \iff |t_i(x) - t_j(x)| \leq 1$.

It turns out that the same relationship existing between implications and functional dependencies, also exists between approximate-matching dependencies and implications in this new generalization of a binarized formal context as shown in the following main result:

Proposition 2. *Let T be a set of tuples, and $\mathcal{U}$ a set of attributes. The approximate-matching dependency $X \to Y$ $(X, Y \subseteq \mathcal{U})$ holds in T if and only if the implication $X \to Y$ holds in the formal context $\mathbb{K}_\theta = (Pair(T), \mathcal{U}, I_\theta)$.*

Proof. The approximate-matching dependencies $X \to Y$ holds in T if and only if for all pairs of tuples t_i, t_j, we have that $\forall x \in X : t_i \theta_x t_j$ implies that $\forall y \in Y : t_i \theta_y t_j$. The former occurs if and only if $\forall x \in X : ((t_i, t_j), x) \in I_\theta$, while the latter occurs if and only if $\forall y \in Y : ((t_i, t_j), y) \in I_\theta$. It follows that $X \to Y$ holds in $\mathbb{K}_\theta$. ∎

6. Experiments

Characterizations of approximate-matching dependencies through partition pattern structures [12] (Section 5.2), and through standard FCA by the binarization of a dataset (Section 5.3) are said to be "equivalent" as they yield isomorphic concept lattices.

In this section, we experiment with both methods to highlight their strengths and weaknesses. Experiments were performed over an Intel Xeon machine with 6 cores running at 2.27GHz and 32GB of RAM machines. All algorithms were implemented in C++ and compiled with the $-O3$ optimization.

6.1. Dataset Description and Experimental Settings

Experiments were performed over 3 datasets from the UCI machine learning repository and 3 datasets from the JASA data archive, namely the *diagnosis*[1],

[1]`http://archive.ics.uci.edu/ml/datasets/Acute+Inflammations`

contraceptive[2], *servo*[3], *caulkins*[4], *hughes-r*[5], and *pglw00*[6] datasets, described in Table 2.

For the *caulkins* dataset we did not considered the columns with redundant information about the weight of the entry –there were two columns indicating this weight, one with the original information and another with its correspondent gram representation; we used the gram representation– and the column containing the value "gram" for every object. For the *hughes-r* dataset we added four binary values encoding metadata for the first three columns. Additionally, we modeled negative values as "missing data" which in our case means that two tuples are considered dissimilar if they both have "missing data" for a given attribute. Changes to other datasets are just related to the conversion of categorical entries represented with a string to a number.

Datasets are binarized by generating a formal context with $|T|^2$ objects and $|\mathcal{U}|$ attributes. An object in the formal context represents two tuples and its attributes are those for which both tuples have *similar* values. Formal contexts are also *clarified* by fusing objects with the same attributes [11]. Clarification has no effect on the calculation of dependencies but can significantly reduce the size of the formal context. We report results for both non-clarified and clarified formal contexts. Formal concepts were calculated using an implementation of the *AddIntent*[7] algorithm [20] from which approximate-matching dependencies can be characterized.

Pattern structures are generated from each dataset by considering a set of similarity parameters θ_x for each attribute $x \in \mathcal{U}$. Binary and categorical attributes were considered as numerical attributes provided with a θ value of 0. Table 1 shows the different values taken by θ for all datasets. These values were selected arbitrarily.

In order to be fair with the comparison of both approaches, we have modified the same *AddIntent* implementation to process pattern structures. This modification consisted in overriding the computation of description intersections and the subsumption test for any two descriptions. Both operations require $|T|^2$ operations in the worst case scenario. For the sake of efficiency, we used striped descriptions, i.e. we do not keep in memory tolerance blocks that are singletons as in [9] and [12].

6.2. Experimental Results

Table 2 reports on the execution times for building the concept lattices for both, the binarization approach (non-clarified and clarified), and pattern structures. For non-clarified and clarified formal contexts, execution times are reported as a sum of a preprocessing time (binarization/clarification) and the

[2]`http://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice`
[3]`http://archive.ics.uci.edu/ml/datasets/Servo`
[4]`http://lib.stat.cmu.edu/jasadata/caulkins-p`
[5]`http://lib.stat.cmu.edu/jasadata/hughes-r`
[6]`http://lib.stat.cmu.edu/jasadata/pglw00.zip`
[7]`https://github.com/codocedo/sephirot/`

Dataset	θ-values
Diagnosis	$\theta_1 = 0.3$
Contraceptive	$\theta_1 = 5$
Servo	$\theta_3 = 5$
Caulkins	$\theta_1 = 5,\ \theta_5, \theta_{12} = 300,\ \theta_6, \theta_7, \theta_{10} = 2000,\ \theta_8, \theta_9, \theta_{11} = 10$
Hughes-r	$\theta_1, \theta_2, \theta_3 = 3$
Pglw00	$\theta_1 = 5$

Table 1: Similarity parameters (theta values) for each dataset. Parameters not reported are set to zero.

Pattern structures $(M, (D, \sqcap), \delta)$						
Dataset	$\vert M \vert$	$\vert T \vert = \vert G \vert$	#Con.	Num.	Cat.	Exec. Time [s]
Diagnosis	8	120	98	1	7	0.81
Contraceptive	10	1473	1024	1	9	734.2
Servo	5	167	28	1	4	1.30
Caulkins	12	1685	2704	9	3	19783.3
Hughes-r	12	401	754	3	9	24.35
Pglw00	6	17995	64	1	5	55.84
Formal context $(\mathfrak{B}_2(G), M, I)$						
Dataset	$\vert G \vert$	$\vert M \vert$	#Con.	Num.	Cat.	Exec. Time [s]
Diagnosis	7082	8	98	1	7	0.32 + 0.09
Contraceptive	1082307	10	1024	1	9	120.46 + 47.6
Servo	13688	5	28	1	4	0.54 + 0.1
Caulkins	1412827	12	2704	9	3	168.19 + 102.249
Hughes-r	80200	12	754	3	9	5.11 + 3.85
Pglw00	161892017	6	64	1	5	-
Formal context $(\mathfrak{B}_2(G), M, I)$ **clarified**						
Dataset	$\vert G \vert$	$\vert M \vert$	#Con.	Num.	Cat.	Exec. Time [s]
Diagnosis	50	8	98	1	7	0.32 + 0.02
Contraceptive	1017	10	1024	1	9	120.46 + 0.089
Servo	25	5	28	1	4	0.54 + 0.006
Caulkins	1146	12	2704	9	3	168.19 + 0.169
Hughes-r	1054	12	754	3	9	5.11 + 0.063
Pglw00	-	6	64	1	5	-

Table 2: Datasets and execution times (Con. : Concepts. Num. : # Numerical attributes. Cat. : # Categorical attributes. The symbol "-" indicates that the value could not be obtained by computational limitations.

execution time of the *AddIntent* algorithm, respectively. Clarification is performed using a linear hashing approach during the binarization process, thus the pre-processing time for both cases is the same. For pattern structures, execution times consider the time needed for the transformation of the numerical dataset into a pattern structure as well as the calculation of the concept lattice.

For all datasets, with the exception of *pglw00* using binarization showed more efficient than processing the equivalent pattern structure.

An interesting exception occurs with the dataset *pglw00*. Since the dataset contains around 10^4 tuples, its binarization should yield a formal context with more than 10^8 objects (roughly 161 millions). This sheer size of elements makes prohibitive the calculation of the concept lattice derived from both the formal context and the clarified formal context. To illustrate this fact consider that if each object could be represented by only a single integer variable with size 8 bytes, then the whole formal context would have a size of around 1 GB of memory (this would be the best case scenario). This is particularly interesting considering that the formal context contains only 6 attributes, meaning that the concept lattice can contain up to 64 formal concepts.

6.3. Synthesis

Under the evidence shown by the experimental results, we can conclude that the use of pattern structures is of critical importance for mining approximate-matching dependencies in medium-large datasets, where binarization and clarification are not possible due to computational limitations. Nevertheless, for sufficiently small datasets, the evidence shows that using standard FCA is a better option. The scripts and binaries necessary to replicate the experiments are freely available on-line[8].

7. Related Work and Discussion

Dependency theory has been an important subject of database theory for more than thirty years. Several types of dependencies have been proposed, capturing different semantics useful for different applications such as query optimization, normalization, data cleaning, error detection, among others.

FDs are first class citizens for checking the consistency and the quality of a database [21, 4, 22], and as well for guiding the database design [5]. In particular, when considering the web of data and RDF triples, FDs can be used for checking completeness and quality of the data. In [23, 24], authors are studying information integration in the web of data with varying degrees of quality. FDs are used for detecting triples which are erroneous w.r.t. domain knowledge. In the same way, in [25], authors are interested in using implications derived from RDF data, i.e. a special kind of FDs, for checking data completeness in DBpedia. The transformation of RDF data into a pattern structure and the following extraction of implications and their inverse are the basis of this research work. Actually, approximate-matching dependencies could be used for extending this research as they would be very well adapted for dealing with imperfect RDF data. This line of research work in RDF data points out the growing importance of FDs for complex data such as RDF triples and graphs [26].

As already discussed above, standard functional dependencies are not always suitable for managing several tasks and data settings. Accordingly, many generalizations have been proposed over the last decades, which were recently and exhaustively reviewed and classified in [6]. This classification separates (i) *extent relaxations* from (ii) *attribute-comparison relaxations*. In the former, FDs are relaxed on their coverage, e.g. a functional dependency only holds in a subset of the data. In the latter, the notion of agreement between two tuples for a

[8]`https://github.com/codocedo/sephirot/tree/DAM16`

Example 7. *An example based on "Average Daily Temperature Archive" showing the monthly average temperatures for different cities [31].*

id	*Month*	*Year*	*Av. Temp.*	*City*
t_1	*1*	*1995*	*36.4*	*Milan*
t_2	*1*	*1996*	*33.8*	*Milan*
t_3	*5*	*1996*	*63.1*	*Rome*
t_4	*5*	*1997*	*59.6*	*Rome*
t_5	*1*	*1998*	*41.4*	*Dallas*
t_6	*1*	*1999*	*46.8*	*Dallas*
t_7	*5*	*1996*	*84.5*	*Houston*
t_8	*5*	*1998*	*80.2*	*Houston*

given attribute is revisited leading to two main subtypes of dependencies. The equality between two values is relaxed for approximate-matching dependencies while an ordering between values must be respected for ordered dependencies.

The general assumption for extent relaxations is that a FD does not necessarily hold in the whole dataset. A FD $X \to Y$ holds in a table T if and only if it holds "for all" pairs of tuples in T. This is precisely this "for all" condition that should be relaxed, i.e. the extent of the table in which this condition must hold. There are several ways to relax the universal condition. A threshold can be given as a percentage such as "Approximate Dependencies" [27], as an impurity function such as "Purity Dependencies" [28], as a minimal number of tuples for each attribute such as "Numerical Dependencies" [29], or as a probability such as "Probabilistic functional dependencies" [30].

For example, considering the dependency `Month` $\to$ `Av.Temp` in Example 7, we can check that 6 tuples should be removed before verifying the dependency: we keep only one tuple for `Month 1` and one tuple for `Month 5` (actually just as if we remove "duplicates"). Then, if the threshold is equal to or larger than 75%, `Month` $\to$ `Av.Temp` is a valid approximate dependency.

Attribute-comparison relaxations include "Attribute-Relaxing Dependencies" and form the second main group of generalizations of FDs. Here, the relaxation holds on the equality condition, which is replaced with a less constrained condition, e.g. "sufficiently close" according to a set of attributes. Then it is possible to distinguish "approximate-matching dependencies" where the attribute comparison is computed thanks to an approximation function, e.g. a distance function, and "order-like dependencies" where values must be ordered instead of being similar.

In this way, "Neighborhood Dependencies" are defined to express regularities in datasets [32]. Given a FD $X \to Y$ and a tuple t, the value of $t[X]$ *determines* the value of $t[Y]$. Then the value of $t[Y]$ is determined not only by the value of $t[X]$ but also by the *neighbor values* of $t[X]$.

We can illustrate such dependencies thanks to Example 7. Let us define functions θ_{Month} and $\theta_{Av.Temp.}$ as follows:

$$\theta_{Month}(m_1, m_2) = 1 - \frac{min(|m_1 - m_2|, min(m_1, m_2) + 12 - max(m_1, m_2))}{12}$$

$$\theta_{Av.Temp.}(t_1, t_2) = 1 - \frac{|t_1 - t_2|}{max(Av.Temp.) - min(Av.Temp)}$$

These functions can be extended into a predicate such as: $\theta_{Month}(m_1, m_2) \geq 0.5$ and $\theta_{Av.Temp.}(t_1, t_2) \geq 0.5$. Then we can define a neighborhood dependency as follows: $\theta_{Month} \rightarrow \theta_{Av.Temp}$. If the values of the attribute $Month$ in a pair of tuples are *close* enough, then, the values of the attribute $Av.Temp$ are also close.

Furthermore, "Differential Dependencies" are defined to extend the notion of equality in FDs [33] They are used to detect violations or inconsistencies in datasets, to optimize queries, to partition data in parts that are somehow similar or detect duplicates in datasets. They are based on a metric distance where each attribute is associated with a different metric distance depending on the nature of the attribute values. Specific constraints can also be attached to some particular attributes w.r.t. domain knowledge.

For example, a differential function over an attribute $x \in X$ and a pair of tuples t, t' can be defined as a Boolean function that evaluates true when a constraint on the values $\theta(t[X], t'[X])$ holds. Such a constraint can be specified thanks to comparison operators, i.e. $\{=, <, >, \leq, \geq\}$, and a threshold δ, e.g. $\theta(t[x], t'[x]) \leq \delta$. Then each attribute has a metric distance θ and an associated constraint, as for neighborhood dependencies. The differential function θ also induces a relation among the set of tuples according to a set of attributes $X \subseteq \mathcal{U}$ as $\theta_{Month}(m_1, m_2)$ and $\theta_{Av.Temp.}(t_1, t_2)$ were defined above for neighborhood dependencies.

Going back to FCA, we have used tolerance relations for computing FDs in [18, 13]. Tolerance relations provide an elegant support for characterizing FDs and approximate-matching dependencies as well. Actually, implications in a formal context exactly correspond to the set of FDs as discussed in [11, 34]. In addition, the relations with conditional FDs, association rules and FCA are also studied in [35, 36].

Moreover, order-like dependencies are studied in the framework of FCA in [37]. They cannot be directly handled in the present framework, as not only blocks of tolerances should be considered, but also the ordering of the objects in each block should coincide as well [38, 6]. Actually, searching for these ordered dependencies is related to the mining of closed gradual itemsets [39]. A first FCA characterization of such a mining problem is provided in a "kind of" pattern structure of sequential patterns [40].

8. Conclusion

In this paper, we have presented a generalization of functional dependencies, namely approximate-matching dependencies, which accounts for "uncrisp dependencies" in the sense that two attribute values are considered as "equal" as soon as they belong to a given interval, i.e. small variations over attribute values are allowed. We discussed how this family of functional dependencies is relevant and share a main basic feature, i.e. a similarity measure depending on the semantics of each attribute. In addition, we have presented a characterization of this family of dependencies using the formalism of pattern structures extending from previous results that frames functional dependencies in the framework of formal concept analysis. This provides an efficient computational framework acknowledged by a series of experimental studies.

Our major result shows that the underlying model for all approximate-matching dependencies can be formalized using tolerance relations for comparing values between tuple attributes, just as equivalence relations –or equivalently partitions– gives the underlying model for standard functional dependencies.

In our future research work, we will be mainly interested in ordered dependencies and in the use of approximate-matching dependencies in classifying the RDF triples in the web of data.

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) and partially funded by the French National Project FUI AAP 14 Tracaverre 2012-2016.

References

[1] D. Maier, The Theory of Relational Databases, Computer Science Press, 1983.

[2] C. Beeri, M. Dowd, R. Fagin, R. Statman, On the Structure of Armstrong Relations for Functional Dependencies, Journal of the ACM 31 (1) (1984) 30–46.

[3] J. Ullman, Principles of Database Systems and Knowledge-Based Systems, volumes 1–2, Computer Science Press, Rockville (MD), USA, 1989.

[4] W. Fan, F. Geerts, Foundations of Data Quality Management, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2012.

[5] H. Mannila, K.-J. Räihä, The Design of Relational Databases, Addison--Wesley, Reading (MA), USA, 1992.

[6] L. Caruccio, V. Deufemia, G. Polese, Relaxed Functional Dependencies – A Survey of Approaches, IEEE Transactions on Knowledge and Data Engineering 28 (1) (2016) 147–165.

[7] A. Day, The lattice theory of fonctionnal dependencies and normal decompositions, International Journal of Algebra and Computation 02 (04) (1992) 409–431.

[8] J. Demetrovics, G. Hencsey, L. Libkin, I. B. Muchnik, Normal Form Relation Schemes: A New Characterization, Acta Cybernetica 10 (3) (1992) 141–153.

[9] S. Lopes, J.-M. Petit, L. Lakhal, Functional and Approximate Dependency Mining: Database and FCA Points of View, Journal of Experimental and Theoretical Artificial Intelligence 14 (2-3) (2002) 93–114.

[10] N. Caspard, B. Monjardet, The Lattices of Closure Systems, Closure Operators, and Implicational Systems on a Finite Set: A Survey, Discrete Applied Mathematics 127 (2) (2003) 241–269.

[11] B. Ganter, R. Wille, Formal Concept Analysis, Springer, Berlin, 1999.

[12] J. Baixeries, M. Kaytoue, A. Napoli, Computing Functional Dependencies with Pattern Structures, in: L. Szathmary, U. Priss (Eds.), Concept Lattices and Applications, CLA, Vol. 972 of CEUR Workshop Proceedings, CEUR-WS.org, 2012, pp. 175–186.

[13] J. Baixeries, M. Kaytoue, A. Napoli, Characterizing Functional Dependencies in Formal Concept Analysis with Pattern Structures, Annals of Mathematics and Artificial Intelligence 72 (1–2) (2014) 129–149.

[14] B. Ganter, S. O. Kuznetsov, Pattern Structures and Their Projections, in: H. S. Delugach, G. Stumme (Eds.), Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001), LNCS 2120, Springer, 2001, pp. 129–142.

[15] J. Baixeries, M. Kaytoue, A. Napoli, Computing Similarity Dependencies with Pattern Structures, in: M. Ojeda-Aciego, J. Outrata (Eds.), Concept Lattices and Applications, CLA, Vol. 1062 of CEUR Workshop Proceedings, CEUR-WS.org, 2013, pp. 33–44.

[16] M. Agier, J. Petit, E. Suzuki, Unifying Framework for Rule Semantics: Application to Gene Expression Data, Fundamenta Informaticae 78 (4) (2007) 543–559.

[17] S. O. Kuznetsov, Galois Connections in Data Analysis: Contributions from the Soviet Era and Modern Russian Research, in: B. Ganter, G. Stumme, R. Wille (Eds.), Formal Concept Analysis, Foundations and Applications, Lecture Notes in Computer Science 3626, Springer, 2005, pp. 196–225.

[18] M. Kaytoue, Z. Assaghir, A. Napoli, S. O. Kuznetsov, Embedding tolerance relations in formal concept analysis: an application in information fusion, in: CIKM, ACM, 2010, pp. 1689–1692.

[19] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo, Fast Discovery of Association Rules, in: Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996, pp. 307–328.

[20] S. O. Kuznetsov, S. A. Obiedkov, Comparing performance of algorithms for generating concept lattices, Journal of Experimental and Theoretical Artificial Intelligence 14 (2–3) (2002) 189–216.

[21] W. Fan, Dependencies revisited for improving data quality, in: M. Lenzerini, D. Lembo (Eds.), Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada, ACM, 2008, pp. 159–170.

[22] W. Fan, Data quality: From theory to practice, SIGMOD Rec. 44 (3) (2015) 7–18.

[23] Y. Yu, J. Heflin, Extending functional dependency to detect abnormal data in RDF graphs, in: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, E. Blomqvist (Eds.), The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I, Vol. 7031 of Lecture Notes in Computer Science, Springer, 2011, pp. 794–809.

[24] Y. Yu, Y. Li, J. Heflin, 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, 2011, pp. 154–157.

[25] M. Alam, A. Buzmakov, V. Codocedo, A. Napoli, Mining definitions from RDF annotations using formal concept analysis, in: Q. Yang, M. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 823–829.

[26] W. Fan, Y. Wu, J. Xu, Functional dependencies for graphs, in: F. Özcan, G. Koutrika, S. Madden (Eds.), Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, ACM, 2016, pp. 1843–1857.

[27] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen, TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies, Computer Journal 42 (2) (1999) 100–111.

[28] D. A. Simovici, D. Cristofor, L. Cristofor, Impurity measures in databases, Acta Informatica 38 (5) (2002) 307–324.

[29] J. Grant, J. Minker, Normalization and axiomatization for numerical dependencies, Information and Control 65 (1) (1985) 1 – 17.

[30] 12th International Workshop on the Web and Databases, WebDB 2009, Providence, Rhode Island, USA, June 28, 2009.

[31] U. of Dayton, Environmental Protection Agency Average Daily Temperature Archive.

[32] R. Basse, J. Wijsen, Neighborhood Dependencies for Prediction, in: D. Cheung, G. Williams, Q. Li (Eds.), Advances in Knowledge Discovery and Data Mining, Vol. 2035 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001, pp. 562–567.

[33] S. Song, L. Chen, Differential Dependencies: Reasoning and Discovery, ACM Transactions on Database Systems 36 (3) (2011) 16:1–16:41.

[34] J. Baixeries, Lattice Characterization of Armstrong and Symmetric Dependencies (PhD Thesis), Universitat Politècnica de Catalunya, 2007.

[35] R. Medina, L. Nourine, A Unified Hierarchy for Functional Dependencies, Conditional Functional Dependencies and Association Rules, in: S. Ferré, S. Rudolph (Eds.), International Conference on Formal Concept Analysis, ICFCA, Vol. 5548 of Lecture Notes in Computer Science, Springer, 2009, pp. 98–113.

[36] R. Medina, L. Nourine, Conditional Functional Dependencies: An FCA Point of View, in: L. Kwuida, B. Sertkaya (Eds.), International Conference on Formal Concept Analysis, ICFCA, Vol. 5986 of Lecture Notes in Computer Science, Springer, 2010, pp. 161–176.

[37] V. Codocedo, J. Baixeries, M. Kaytoue, A. Napoli, Characterization of Order-like Dependencies with Formal Concept Analysis, in: M. Huchard, S. Kuznetsov (Eds.), Proceedings of the Thirteenth International Conference on Concept Lattices and Their Applications, CLA 2016, Moscow, Russia, July 18-22., Vol. 1624 of CEUR Workshop Proceedings, CEUR-WS.org, 2016, pp. 123–134.

[38] W. Ng, Ordered functional dependencies in relational databases, Information Systems 24 (7) (1999) 535 – 554.

[39] T. D. T. Do, A. Termier, A. Laurent, B. Négrevergne, B. O. Tehrani, S. Amer-Yahia, PGLCM: efficient parallel mining of closed frequent gradual itemsets, Knowledge and Information Systems 43 (3) (2015) 497–527.

[40] S. Ayouni, A. Laurent, S. B. Yahia, P. Poncelet, Mining Closed Gradual Patterns, in: L. Rutkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, J. M. Zurada (Eds.), Artificial Intelligence and Soft Computing, 10th International Conference, ICAISC 2010, Zakopane, Poland, June 13-17, 2010, Part I, Vol. 6113 of Lecture Notes in Computer Science, Springer, 2010, pp. 267–274.