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

Jaume Baixeries, Victor Codocedo, Mehdi Kaytoue, Amedeo Napoli

Elsevier Editorial System(tm) for Discrete Applied Mathematics

Manuscript Draft

Manuscript Number:

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

Article Type: SI: CLA'14

Keywords: Pattern Structures
Formal Concept Analysis
Data Dependencies
Knowledge Discovery

Corresponding Author: Dr. Jaume Baixeries,

Corresponding Author's Institution:

First Author: Jaume Baixeries

Order of Authors: Jaume Baixeries; Mehdi Kaytoue; Victor Codecedo; Amedeo Napoli

Abstract: Functional dependencies (FDs) provide valuable knowledge on the relations between the attributes of a data table.
A functional dependency holds when the values of an attribute can be determined by another.
It is 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 are 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 to uncrisp the basic notion of FDs, and can be applied in many different fields, e.g. data quality, data mining, behavior analysis, data cleaning or data partition...
Here we show that these dependencies can be formalized in the framework of Formal Concept Analysis (FCA).
Such a formalization was previously introduced for basic FDs, but needs to be adapted and extended for approximate-matching dependencies.
Our new result states 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 adapt basic FCA algorithms to construct a pattern concept lattice that entails these dependencies after a slight and tractable transformation of the original data.

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

Jaume Baixeries[a] , Victor Codocedo[c], Mehdi Kaytoue[c] 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*

Abstract

Functional dependencies (FDs) provide valuable knowledge on the relations between the attributes of a data table. A functional dependency holds when the values of an attribute can be determined by another. It is 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 are 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 to uncrisp the basic notion of FDs, and can be applied in many different fields, e.g. data quality, data mining, behavior analysis, data cleaning or data partition... Here we show that these dependencies can be formalized in the framework of Formal Concept Analysis (FCA). Such a formalization was previously introduced for basic FDs, but needs to be adapted and extended for approximate-matching dependencies. Our new result states 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 adapt basic FCA algorithms to construct a pattern concept lattice that entails these dependencies after a slight and tractable transformation 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 of a database but also to guide the database design [4].

However, the definition of FDs is too strict for several useful tasks, for instance, when one should deal with imprecision in the data, 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 [5]. According to this classification, two main strategies are presented: "extent relaxation" and "attribute relaxation" (in agreement with the terminology introduced in [5]).

Characterizing and computing FDs are strongly related to lattice theory. For example, lattice characterizations of a set of FDs are studied in [6, 7, 8, 9]. Following the same line, a characterization of FDs within Formal Concept Analysis (FCA) is proposed in [10]. In the last case, the original data table is turned into a formal context (i.e. binary table) and implications of this context are in one-to-one correspondence with the set of FDs. However, the resulting formal context has a quadratic number of objects w.r.t. the original dataset. To avoid this, [11] and [12] show how to use pattern structures, introduced in FCA by [13]. Moreover, in [14] 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 farther in the present paper and as well in [15]. In the later, a unifying framework in which any "well-formed" semantics for rules may be integrated is introduced. In the same way, this is also what we try to define in this paper, for generalizations of FDs and in the framework of FCA and Pattern Structures.

This paper presents an extended and updated version of [14], 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 only, we extend here the characterization to the family of approximate-matching dependencies, based on pattern structures and symmetric relations in pattern structures [16]. We also show that the classical FCA algorithms can be –almost directly– applied to compute similarity dependencies.

The 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 symmetric relations and we show how the dependencies that are enumerated in Section 3 are based on symmetric 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 similarity dependencies with pattern structures.

2. Notation and Functional Dependencies

2.1. Notation

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 then a table T is a set of tuples. Usually a table is presented as a matrix, as in the table of Example 1, where the set of tuples (or objects) is $T = \{t_1, t_2, t_3, t_4\}$ and $\mathcal{U} = \{a, b, c, d\}$ is the set of attributes. Sometimes the set notation is omitted and we write ab instead of $\{a, b\}$.

The functional notation allows to associate an attribute with its value. We define the functional notation of a tuple for a set of attributes X as follows, assuming that there exists a total ordering on $\mathcal{U}$. 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. *This is an example of a table $T = \{t_1, t_2, t_3, t_4\}$, based on the set of 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 are also dealing with the set of partitions of a set. Let S be any arbitrary finite set, then, Part(S) is the set of all possible partitions that can be formed with S. When equipped with an appropriate partial ordering, Part(S) is a lattice [17]. Moreover, partitions can also be considered as equivalence classes induced by an equivalence relation.

2.2. Functional Dependencies

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 $t_2(c) \neq t_4(c)$.

There is an alternative way of considering FDs using partitions of the set of tuples T. Taking a set of attributes $X \subseteq \mathcal{U}$, we define the partition of tuples induced by this set as follows.

Definition 2. *Let $X \subseteq \mathcal{U}$ be a set of attributes in a table T. The partition of T induced by X is a set of equivalence classes $\Pi_X(T) = \{c_1, c_2, \ldots, c_m\}$ such that, for all $c_k \in \Pi_X(T)$:*

$$\forall t_i, t_j \in c_k : t_i[X] = t_j[X]$$

For example, if we consider the table in Example 1, we have $\Pi_a(T) = \{\{t_1, t_3\}, \{t_2, t_4\}\}$.

Given X, $\Pi_X(T)$ is a partition of the set of tuples T, which, alternatively, induces an equivalence relation. Then we have:

1. $\bigcup \Pi_X(T) = T$, for all $X \subseteq \mathcal{U}$.
2. $c_i \cap c_j = \emptyset$ for all $c_i, c_j \in \Pi_X(T)$, $i \neq j$.

The classes in a partition induced by X are disjoint and they cover all the tuples in T. The set of all partitions of a set T is $\mathrm{Part}(T)$. We define the following order relation on the set $\mathrm{Part}(T)$:

$$\forall P_i, P_j \in \mathrm{Part}(T) : P_i \leq P_j \iff \forall c \in P_i : \exists c' \in P_j : c \subseteq c'$$

For example: $\{\{t_1\}, \{t_2\}, \{t_3, t_4\}\} \leq \{\{t_1\}, \{t_2, t_3, t_4\}\}$. As a consequence, we have that $\forall X, Y \subseteq \mathcal{U}$:

$$X \subseteq Y \text{ implies } \Pi_X(T) \geq \Pi_Y(T)$$

According to the partitions induced by a set of attributes, we have an alternative way of defining the necessary and sufficient conditions for a functional dependency to hold:

Proposition 1 ([18])**.**
A functional dependency $X \to Y$ holds in T if and only if $\Pi_X(T) = \Pi_{X \cup Y}(T)$.

Since $\forall X, Y \subseteq \mathcal{U}$ the relation $\Pi_X(T) \geq \Pi_{X \cup Y}(T)$ holds, this proposition can be rephrased as follows: a functional dependency $X \to Y$ holds in T if and only if $\Pi_X(T) \leq \Pi_{X \cup Y}(T)$.

Again, taking the table in Example 1, we have that $a \to d$ holds and that $\Pi_a = \Pi_{ad}$ since $\Pi_a(T) = \{\{t_1, t_3\}, \{t_2, t_4\}\}$ and $\Pi_{ad}(T) = \{\{t_1, t_3\}, \{t_2, t_4\}\}$ (actually $d \to a$ holds too).

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. 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 are not meaning *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 every individual, it appears that sometimes SSN is shared by more than one individual. In such a case, the FD ``social security number → 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 → name, surname'' does not hold, would not give us any relevant information.

In some other cases, FDs may fail in providing us interesting information. 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 X` 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 Y` for example. These few cases prevent the FD `beer brand` → `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 of FDs have been introduced. These generalizations can be divided into two main groups [5], depending on the strategy to relax the semantics of FDs. These two categories are (1) functional dependencies that relax their extent, and (2) functional dependencies that relax the condition on the attributes.

3.1. Extent-Relaxing Dependencies

The general assumption in this category is that a FD does not necessarily hold in the whole dataset. A FD $X \rightarrow Y$ holds in a table T if and only if it holds for all pairs of tuples in T. This is precisely the "for all" condition that should be relaxed, i.e. the extent of the table in which this condition must hold. An *extent-relaxed* FD holds in a table T if and only if it holds in a fraction of all the pairs of tuples in T. There are many different ways to relax the "universal" condition. A threshold can be given as a percentage (e.g. Approximate Dependencies [18]), as an impurity function (e.g. Purity Dependencies [19]), as

a number of tuples for each attribute (e.g. Numerical Dependencies [20]), or as a probability (e.g. Probabilistic functional dependencies [21]). Let us consider "Approximate Dependencies" as a paradigmatic example.

Example 2. *Excerpt of the Average Daily Temperature Archive showing the monthly average temperatures for different cities [22].*

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*

Approximate Dependencies [18] can be likened to *association rules* [23]. The validity of association rules depends on certain metrics such as confidence and support. Confidence measures the proportion of the whole dataset in which an association rule holds. An association rule R having a confidence of 100% is in fact an implication. Another way of interpreting the semantics of confidence is the following: when an association rule has a confidence of x%, then removing the $100-x$% of the tuples which are not in agreement allows the association rule to become an implication. This idea of counting the number of tuples in which a given dependency holds, or, more precisely, the minimal number of tuples to be removed to allow a FD to hold, is the main idea underlying "Approximate Dependencies".

In a table, tuples that prevent a FD from holding can be seen as exceptions (or errors) for that dependency, since their removal would allow the dependency to hold. Then a threshold can be set to define a set of approximate dependencies holding in the table. For example, a threshold of 10% means that all FDs holding after removing up to 10% of the tuples of a table are valid approximate dependencies. The set of tuples to be removed for validating a FD does not need to be the same for each approximate dependency.

Considering in Example 2 the dependency `Month` $\rightarrow$ `Av. Temp`, 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` $\rightarrow$ `Av. Temp` is a valid approximate dependency.

3.2. Attribute-Relaxing Dependencies

"Attribute-Relaxing Dependencies" form the second main group of generalizations of FDs, where the relaxation holds on the equality condition. 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.

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. order-like dependencies can be studied in the framework of FCA as shown in [24]. However, for the sake of simplicity and clarity we will not discuss further these dependencies in the present paper.

Attribute-relaxing dependencies are present in the database literature. An extension of Codd's relational model in which attribute domains are equipped with similarity relations is presented in [25]. Fuzzy attribute implications in FCA are also related with this sort of dependencies, as fully explained in [26].

In the remainder of this section, we detail so-called "Neighborhood Dependencies" and "Differential Dependencies".

3.2.1. Neighborhood Dependencies

Neighborhood Dependencies (NDs) were defined to express regularities in datasets [27]. Given a FD $X \to Y$ and a tuple t, the value of $t[X]$ *determines* the value of $t[Y]$. With neighborhood dependencies, the value of $t[Y]$ is *predicted* not only by the value of $t[X]$ but also by the *neighbor values* of $t[X]$.

Let T be a set of tuples and $x \in \mathcal{U}$. A *closeness function* on the attribute x is a function $\theta_x : T[x] \times T[x] \mapsto [0, 1]$ that computes how near are two values of the attribute x in a scale $[0, 1]$, where the closer to 1 the function evaluates for two values, the more similar they are (and vice-versa). Obviously, $\theta_x(a, a) = 1$.

Let $\delta \in [0, 1]$ be a threshold, and let $a, b \in T[x]$. Then, $\theta_x(a, b) \geq \delta$ is a *neighborhood predicate* on the attribute x. We still use the same notation θ_x to indicate a neighborhood predicate, where the threshold is tacitly assumed. We suppose that each attribute $x \in \mathcal{U}$ has a related threshold δ and an associated comparison operator which may be, obviously, different for each attribute. This predicate induces a relation on the set of tuples T w.r.t. θ and according to a set of attributes $X \subseteq \mathcal{U}$.

Pair of tuples $t, t' \in T$ are said to be related according to a set of attributes $X \subseteq \mathcal{U}$, denoted by $(t\ \theta_X\ t')$, if:

$$t\theta_X t' \Leftrightarrow \bigwedge_{x \in X} \theta_x(t[x], t'[x])$$

A neighborhood dependency $\theta_X \to \theta_Y$ holds in a dataset T iff:

$$\forall t, t' \in T : \bigwedge_{x \in X} \theta_x(t[x], t'[x]) \to \bigwedge_{y \in Y} \theta_y(t[y], t'[y])$$

This can be rewritten as:

$$\forall t, t' \in T : t\ \theta_X t' \rightarrow t\ \theta_Y t'$$

This dependency holds for each pair of tuples t, t', if all the neighborhood predicates for the attributes $x \in X$ evaluate to true, then all the neighborhood predicates for the attributes $y \in Y$ evaluate to true as well. We illustrate such dependencies thanks to Example 2. We define the functions θ_{Month} and $\theta_{Av.Temp.}$ as follows:

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

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

They are closeness functions, since they return values in $[0, 1]$, and, given the same values they returns 1, and distant values they returns 0. We now equip these functions with an extra predicate:

$$\theta_{Month}(m_1, m_2) = 1 - \frac{min(|m_1 - m_2|, min(m_1, m_2) + 12 - max(m_1, m_2))}{12} \geq 0.5$$

$$\theta_{Av.Temp.}(t_1, t_2) = 1 - \frac{|t_1 - t_2|}{max(Av.Temp.) - min(Av.Temp)} \geq 0.5$$

In both cases, we have a Boolean predicate and we can define a neighborhood dependency as follows:

$$\theta_{Month} \rightarrow \theta_{Av.Temp}$$

If the value of the attribute $Month$ in a pair of tuples is *close* enough, then, the value of the attribute $Av.Temp$ is also close.

3.2.2. Differential Dependencies

Differential Dependencies were defined to extend the notion of equality in FDs [28] They can be 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.

Differential Dependencies are based on a metric distance and a constraint. A metric distance is a function θ over two values that satisfies the triangle inequality as well as the identity of indiscernibles ($\theta(a, b) = 0 \Leftrightarrow a = b$). Within the context of differential dependencies, each attribute is associated with a different metric distance, that depends on the nature of the attribute values.

A *differential function* over an attribute $x \in X$ and a pair of tuples t, t' is a Boolean function that evaluates true when a constraint on the values $\theta(t[X], t'[X])$ holds. A constraint can be specified by the comparison operators $=, <, >, \leq, \geq$ and a threshold δ, e.g. $\theta(t[x], t'[x]) \leq \delta$ (also simply rewritten as $\theta_x(t, t')$ where the constraint is implicit). Thus, 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 with neighborhood dependency (previous subsection).

For example, let us define:

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

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

We now equip these functions with an extra predicate: θ_{Month} as

$$\theta_{Month}(m_1, m_2) = min(|m_1 - m_2|, min(m_1, m_2) + 12 - max(m_1, m_2)) \leq 4$$

$$\theta_{Av.Temp.}(t_1, t_2) = |t_1 - t_2| \leq 10$$

These two functions a differential dependency such as $\theta_{Month} \rightarrow \theta_{Av.Temp}$, whose semantics is similar to the associated neighborhood dependency in the preceding section, i.e. if the values $Month$ are similar, then the values of $Av.Temp$ are also similar.

4. Characterization of Attribute-Relaxing Dependencies

In this section, we propose a minimal characterization of the attribute-relaxation dependencies introduced in Section 3.2. This characterization will be sufficient for representing these dependencies within the formalism of Pattern Structures.

4.1. Symmetric Relations and Blocks of Tolerance

Firstly, we define a *tolerance* relation in a set S and then the associated *blocks of tolerance*.

Definition 3. *A tolerance relation $\theta \subseteq S \times S$ on a set S is a reflexive (i.e. $\forall s \in S : s\theta s$) and symmetric (i.e. $\forall s_i, s_j \in S : s_i \theta s_j \iff s_j \theta s_i$) relation.*

In [14] we have used *tolerance relations* to compare attribute values. In the following, we will be more restrictive and use a symmetric relation θ for the same purpose. Indeed, in some cases it can be interesting to consider relations which are not reflexive. Then a symmetric relation induces *blocks of tolerance* as follows.

Definition 4. *Given a set S, a subset $K \subseteq S$ and a symmetric relation $\theta \subseteq S \times S$, K is a* block of tolerance *of θ if:*

1. $\forall x, y \in K : x\theta y$ *(pairwise correspondence)*
2. $\forall z \notin K, \exists u \in K : \neg(z\theta u)$ *(maximality)*

Thus we have:

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

Then, we define a partial ordering on the set of all possible symmetric relations in a set S as follows:

Definition 5. *Let θ_1 and θ_2 two symmetric relations in the set S. We say that $\theta_1 \leq \theta_2$ if and only if $\forall K_i \in S/\theta_1 : \exists K_j \in S/\theta_2 : K_i \subseteq K_j$*

This relation is a partial ordering and induces a lattice where the meet and join operations of two symmetric relations θ_1 and θ_2, or, equivalently, on the sets of blocks of tolerance of θ_1 and θ_2 are:

Definition 6. *Let θ_1 and θ_2 two symmetric relations in the set S.*
$\theta_1 \wedge \theta_2 = \theta_1 \cap \theta_2 = \max_S(\{k_i \cap k_j \mid k_i \in S/\theta_1, k_j \in S/\theta_2\})$
$\theta_1 \vee \theta_2 = \theta_1 \cup \theta_2 = \max_S(S/\theta_1 \cup S/\theta_2)$

$max_S(.)$ returns the set of maximal subsets w.r.t. inclusion.

An example of a symmetric relation is the *similarity* that can be defined within a set of integer values as follows. Given two integer values v_1, v_2 and a threshold ϵ (user-defined): $v_1 \theta v_2 \iff |v_1 - v_2| \leq \epsilon$. For example, when $S = \{1, 2, 3, 4, 5\}$ and $\epsilon = 2$, then $S/\theta = \{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}\}$. S/θ is not a partition, as θ is not transitive.

4.2. Tolerance Blocks on Attribute Values

Given a set of tuples T and a set of attributes M, for each attribute $m \in M$, we define a symmetric relation θ_m on the values of m. The set of tolerance blocks induced by θ_m is denoted by T/θ_m. All the tuples in a tolerance block $K \in T/\theta_m$ are *similar* one to the other according to their values w.r.t. m.

Example 3. *Let us define a symmetric relation 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 $\epsilon = 1$ in Example 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 also extend this definition to sets of attributes. Given $X \subseteq \mathcal{U}$, the similarity relation θ_X is defined as follows:

$$(t_i, t_j) \in \theta_X \iff \forall m \in X : (t_i, t_j) \in \theta_m$$

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. Symmetric Relations and Functional Dependencies Relaxing Attribute Comparison

Below we show that relations existing between tuples in Neighborhood, Differential and Similarity Dependencies are, in fact, symmetric relations. In order to prove that, we only need to show that symmetry is met by all these relations.

Neighborhood Dependencies are based on a *closeness function* $\theta_x : T[x] \times T[x] \mapsto \{0, 1\}$ which is symmetric, i.e. $\theta_x(a, b) = \theta_x(b, a)$. Therefore, the composition of this function with a predicate, such as $\theta_x(a, b) \leq \delta$ is also symmetric.

Differential Dependencies are based on a metric distance ($\phi(a, a) = 0$ and $\phi(a, b) = \phi(b, a)$, $\phi(x, y) \in [0, 1]$) and a constraint which can be a comparison operator $\leq, \geq, =$. For instance, a valid constraint would be $\phi(a, b) \leq \delta$. In this case, symmetry holds since metric distances are symmetric and their composition with a constraint is symmetric as well.

4.4. Approximate-Matching Dependencies

Definition 7. *Let $X, Y \subseteq \mathcal{U}$: $X \to Y$ is an approximate-matching dependency iff: $\forall t_i, t_j \in T : t_i \theta_X t_j \to t_i \theta_Y t_j$*

While a 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 4. *We revisit the table in Example 1 and we define the symmetric 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 $t_i(bcd) = t_j(bcd)$.*

Example 5. *Attribute-Relaxing Dependencies having attribute Av. Temp. in their right-hand side from example 2.*

Dependency	*Holds*
`Month -> Av. Temp`	*N*
`Month, Year -> Av. Temp`	*N*
`Month, City -> Av. Temp`	*Y*
`Year -> Av. Temp`	*N*
`Year, City -> Av. Temp`	*N*
`City -> Av. Temp`	*N*

The only approximate-matching dependency that holds is `Month, City` $\to$ `Av. Temp`*, using the following similarity measures for each attribute:*

- $x \ \theta_{Month} \ y \iff |x - y| \leq 0$
- $x \ \theta_{Year} \ y \iff |x - y| \leq 0$

- $x\ \theta_{City}\ y \iff distance(x,y) \leq 500$
- $x\ \theta_{Av.Temp}\ y \iff |x-y| \leq 10$

The relation imposes that the month and year must be the same, whereas the distance between cities should be less than 500 *km and the difference between average temperatures should be less than* 10 *degrees (it should be noticed that all these values are arbitrary). In particular, considering the tuples* t_1, t_2*:*

- $t_1 \theta_{Month,City} t_2$ *since* $t_1(Month) = t_2(Month) = \langle\, 1 \,\rangle$ *and* $t_1(City) = t_2(City) = \langle\, Milan \,\rangle$.
- *From the other side, we have that* $t_1 \theta_{Av.Temp.} t_2$ *since* $|36.4 - 33.8| \leq 10$.

5. Computing Attribute-Relaxing Dependencies with Pattern Structures

5.1. A Brief Introduction to Pattern Structures in FCA

A "Pattern Structure" allows one to apply FCA directly on non-binary data [13]. 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 $d \in (D, \sqcap)$:

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

These operators form a Galois connection between $(\wp(G), \subseteq)$ and $(D, \sqcap)$. Pattern concepts of $(G, (D, \sqcap), \delta)$ are pairs of the form (A, d), $A \subseteq G$, $d \in (D, \sqcap)$, 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, as it is shown in [12]. Firstly, the description of an attribute $m \in M$ is given by $\delta(m) = G/\theta_m$ which is given by the set of tolerance blocks w.r.t. θ_m and $G = T$. As symmetric relations can be ordered (see Definitions 5 and 6) then descriptions can be ordered within a lattice.

A dataset can be represented as a pattern structure $(M, (D, \sqcap), \delta)$ where M is the set of original attributes, and $(D, \sqcap)$ is the set of sets of blocks of tolerance over G provided with the meet operation introduced in Definition 6.

An example of concept formation is given as follows. Consider the table in Example 1. Starting from the set $\{a, c\} \subseteq M$ and assuming that $t_i \theta_m t_j \iff |t_i(m) - t_j(m)| \leq 2$ for all attributes:

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

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 M:

Proposition 2. *An approximate-matching dependency $X \to Y$ holds in a table T if and only if: $\{X\}^{\square} = \{XY\}^{\square}$ in the pattern structure $(M, (D, \sqcap), \delta)$.*

Proof. First of all, we notice that $(t, t') \in \{X\}^{\square}$ if and only if $t(X)\theta_X t'(X)$ which is the same as $\forall x \in X : t(x)\theta_x t'(x)$. We also notice that $\{X, Y\}^{\square} \subseteq \{X\}^{\square}$.
($\Rightarrow$) We prove that if $X \to Y$ holds in T, then, $\{X\}^{\square} = \{X, Y\}^{\square}$, actually $\{X\}^{\square} \subseteq \{X, Y\}^{\square}$. We take an arbitrary pair $(t, t') \in \{X\}^{\square}$, i.e. $t(X)\theta_X t'(X)$. Since $X \to Y$ holds, it implies that $t(XY)\theta_{XY} t'(XY)$, and this implies that $(t, t') \in \{X, Y\}^{\square}$.
($\Leftarrow$) We take an arbitrary pair $t, t' \in T$ such that $t(X)\theta_X t'(X)$.
Therefore, we have that $(t, t') \in \{X\}^{\square}$, and by hypothesis, $(t, t') \in \{XY\}^{\square}$, i.e. $t(XY)\theta_{XY} t'(XY)$. Since this is for all pairs $t, t' \in T$ such that $t(X) = t'(X)$, this implies that $X \to Y$ holds in T. ∎

This proposition is structurally the same that was used in [12] to prove the characterization of functional dependencies with pattern structures. In this case, we have changed and *relaxed* the equality condition by the symmetric relation θ.

5.3. Computing Attribute-Relaxing Dependencies with Binarization in FCA

In [11] it is shown how a *binarization* of a table dataset can be defined. This process consists in creating a formal context such that the set of attributes is the same as that of the dataset ($\mathcal{U}$), whereas the set of objects is the set of all pairs of tuples $Pair(T) = \{t_i, t_j \mid i < j\}$. We have that $((t_i, t_j), m) \in I$ (for all $m \in \mathcal{U}$) if and only if $t_i(x) = t_j(x)$. In Figure 1 we have the example of a dataset (left) and this binarization (middle). The corresponding formal

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 $(\mathcal{B}_2(G), M, I)$ (middle), and the generalization of the binarized formal context, taking $\theta \leq |\ t_i(m) - t_j(m)\ |$.

context is, then, $\mathbb{K} = (Pair(T), \mathcal{U}, I)$. Binarization is also defined in [2] and [4] as *agree sets*, which are pairs of tuples that agree in all the values of a given set of attributes, and disagree in all the rest of them.

This binarization stated that an *implication* $X \rightarrow Y$ holds in the formal context $\mathbb{K} = (Pair(T), \mathcal{U}, I)$ if and only if the functional dependency $X \rightarrow Y$ holds in T. Note that in this case, since $X, Y \subseteq \mathcal{U}$, both the implication $X \rightarrow Y$ and the functional dependency $X \rightarrow Y$ are identical, but they have different semantics.

In the context of similarity dependencies, we generalize this notion of binarization according to a threshold measure θ. In this case, we have the same set of objects and attributes, but we slightly adapt this new relation I_θ as follows: $((t_i, t_j), m) \in I_\theta$ (for all $m \in \mathcal{U}$) if and only if $\theta \leq \mid t_i(m) - t_j(m) \mid$. The resulting context $\mathbb{K}_\theta = (Pair(T), \mathcal{U}, I_\theta)$ is illustrated in Figure 1 (right).

It turns out that the same relationship that exists between implications and functional dependencies, also exists between similarity dependencies and implications in this new generalization of a binarized formal context. This means that the implication $X \rightarrow Y$ holds in the formal context $\mathbb{K}_\theta = (Pair(T), \mathcal{U}, I_\theta)$ if and only if the similarity dependency $X \rightarrow Y$ holds in the dataset T. The proof is quite straightforward:

Proposition 3. *Let T be a dataset, and $\mathcal{U}$ the set of its attributes. The similarity dependency $X \rightarrow Y$ ($X, Y \subseteq \mathcal{U}$) holds in T if and only if the implication $X \rightarrow Y$ holds in the formal context $\mathbb{K}_\theta = (Pair(T), \mathcal{U}, I_\theta)$.*

Proof. The similarity dependency $X \rightarrow Y$ holds in T if and only if for all pairs of tuples t_i, t_j, we have that $\forall m \in X : \theta \leq \mid t_i(m) - t_j(m) \mid$ implies that $\forall m \in Y : \theta \leq \mid t_i(m) - t_j(m) \mid$, if and only if $\forall m \in X : ((t_i, t_j), m) \in I_\theta$ implies that $\forall m \in Y : ((t_i, t_j), m) \in I_\theta$, iff the implication $X \rightarrow Y$ holds in $\mathbb{K}_\theta$. ∎

6. Experiments

In the literature, pattern structures have been defined over various description spaces (partitions, intervals, graphs, etc., see [29]). When an "equivalent" data representation can be given by a formal context, one needs to understand which one is the best in terms of efficiency. By "equivalent" data representation, we mean that they derive isomorphic concept lattices. For example, starting from a numerical dataset, [30] shows that n-dimensional closed intervals can be characterized in two different ways: (i) with a pattern structure having a meet-semi-lattice of intervals as a description space, and (ii) with a formal context obtained with an interordinal scaling. Depending on the original dataset characteristics (number of tuples, number of attributes, distribution of attribute domains, etc.), one data representations is preferred to the other. The aim of the present section is to provide discussion elements on the feasibility and scalability of extracting similarity dependencies with pattern structures and from an "equivalent" derived formal context.

Indeed, we presented how similarity dependencies can be characterized from a formal context built thanks to the binarization of a dataset (yielding a quadratic number of objects). We also showed that partition pattern structures [11] can be adapted to produce a concept lattice isomorphic to the one raised from binarization. Thus we have here two ways for characterizing similarity dependencies with FCA. In this section, we experiment with both methods to highlight their strengths and weaknesses. We used an Intel Xeon machine with 6 cores running at 2.27GHz and 32GB of RAM machines, and all algorithms are implemented in C++ and compiled with the $-O3$ optimization.

6.1. Dataset Description and Experimental Settings

We experimented with 3 datasets from the UCI machine learning repository and 3 datasets from the JASA data archive, namely the *diagnosis*[1], *contraceptive*[2], *servo*[3], *caulkins*[4], *hughes-r*[5], and *pglw00*[6] datasets, described in Table 2.

With the exception of *caulkins* and *hughes-r*, all datasets were used without modification. In *caulkins*, some of the columns were ignored. Specifically, 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. In *hughes-r*, we added four columns to encode the information of the first three columns. All additional columns are binary. Furthermore, the value -1 was used to indicate "empty value", meaning that this information should not be considered, i.e. for the pattern intents generated for the three first columns, objects with the value -1 are not present in any component. Changes to other datasets are just related to the conversion of categorical entries represented with a string to a number.

The discretization procedure to turn a dataset into a binary formal context is achieved via a simple script: for any two objects in T, it gives the attributes from M for which those objects agree, i.e. have similar values. Moreover, a context can be clarified, i.e. the same rows and columns are fused in a fused respectively in a unique row and column [10]. Clarification has no effect on the calculation of dependencies but can significantly reduce the size of the formal context. However, both non-clarified and clarified contexts were used in the experiments. An implementation of the *addIntent*[7] algorithm [31] is used to build the concept lattice from which similarity dependencies can be characterized.

To proceed with pattern structures, each attribute $m \in M$ of the original dataset is described by tolerance blocks of objects from T which depends on the

[1] http://archive.ics.uci.edu/ml/datasets/Acute+Inflammations
[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://code.google.com/p/sephirot/

chosen similarity parameter θ_m. Thanks to the genericity of pattern structures, and to be fair in the algorithmic comparison of the two approaches, we modified the same *addIntent* algorithm implementation to process a pattern structure. The modification consists in overriding the computation of description intersections, and the subsumption test for any two descriptions. Both operations are quadratic w.r.t number of original objects T in the worst case. For the sake of efficiency, we use striped descriptions, i.e. we do not keep in a concept intent the tolerance blocks that are singletons, as in [8] and [11].

Finally, it should be realized that the difference between using symmetric relations or using partitions as attribute descriptions $\delta(m)$ (as presented in [11]) is that tolerance blocks are not restricted to have an empty intersection, i.e. they can overlap. This has a direct impact on the computing efficiency for calculating the concept lattice, as we will see later in this section. Parallelization using the OpenMP[8] library was used to calculate tolerance block intersections improving the efficiency of the lattice building algorithm.

6.2. Experimental results

We process each dataset as follows: (i) with the standard *addIntent* algorithm we build the *concept lattice* of the derived formal context ; (ii) with a *modified addIntent* algorithm we build the *pattern concept lattice* of the pattern structure. Table 2 reports the execution times for building those lattices. For non-clarified and clarified formal contexts, execution times report a sum of the binarization/clarification time and the execution of the AddIntent algorithm, respectively. For pattern structures, the execution times take into account the transformation of the numerical dataset as a pattern structure as well as its processing. It should be noticed that that for the six datasets, there are different number of numerical and categorical attributes. The similarity parameter is used only for both kind of attributes, however for categorical attributes, θ-values are always 0 (equality). Table 1 shows the different values taken by θ for all the datasets. An important observation is that θ-values were selected arbitrarily with no regard for its actual application meaning, but only considering computational purposes.

In all the chosen datasets (with the exception of *pglw00*), processing the formal contexts is more efficient than processing the equivalent pattern structure. Formal context clarification takes the same time as the non-clarified formal context building: when a pair of objects $g \in \mathfrak{B}_2(G)$ is generated, we check that we did not already generate another pair $h \in \mathfrak{B}_2(G)$ such that $g' = h'$. In that case, the pair g is dropped. This is why the pre-processing times for formal contexts and clarified formal contexts are the same in Table 2. The processing of the clarified formal context is the more efficient by far given the great reduction of objects it performs in the formal context. For example, for the *contraceptive* dataset, it reduces the number of pairs from 1 million to 1 thousand. A similar change can be then observed for execution times of the concept lattice

[8] `http://openmp.org/`

construction. The density of the clarified formal contexts is similar through all the datasets (around 50%). This shows that the differences in calculation time are not due to the amount of information present in each formal context. This is further discussed at the end of this section.

6.2.1. Computing with Pattern Structures Can Be Very Efficient

An interesting exception occurs with the dataset *pglw00*. It contains 17995 (10^4) objects meaning that the creation of the formal context should contain more than 10^8 objects (roughly 161 millions). This sheer size of elements makes prohibitive the calculation of the formal context and the clarified formal context. For example, consider that each object can be represented by only a single integer variable with size 8 bytes, then the whole context would have a size of around 12 GB of memory (best case). This is particularly interesting considering that the formal context contains only 6 attributes, meaning that the concept lattice can only contain up to 64 formal concepts (which it does). Through the use of pattern structures we can obtain the formal concepts of *pglw00* in less than a minute and the size of the concept lattice in a compact notation is 13 MB of memory. This makes clear that, while the use of binarization and clarification of the dataset is very useful for small datasets, for slightly larger ones (consider that *pglw00* is only 1 order of magnitude larger than *caulkins* or *contraceptive*), this technique is no longer feasible.

In our previous work, we showed that functional dependencies which are similarity dependencies with $\theta = 0$ can be characterized either with a formal context or a partition pattern structure. It was clear that pattern structures were more efficient for finding functional dependencies. In the present work, this result is less straightforward. The main reason is the following: when dealing with functional dependencies, patterns correspond to partitions, and an object can belong to only one block of the partition. For similarity dependencies, an object can belong to several blocks of tolerance (components). Table 3 gives a few statistics about pattern intents of the pattern concept lattices. For each dataset it shows the average (and standard deviation) of the number of elements per component, as well as the average of the number of tolerance blocks.

6.2.2. The Importance of Numerical Attributes

For the dataset *caulkins* we can see that, in average, a pattern intent contains 551 components. This means that, in average, we should make more than 300K set intersection computations to obtain a single closure. This explains the fact that, even with parallelization, the calculation of the *caulkins* dataset concept lattice using partition pattern structures takes over 5 hours. On the other side, *pgwl00*, while containing 10 times more objects, has an average of 88 components per intent meaning around 8K intersections per closure computation. This is due to the fact that *pgwl00* only has one numerical attribute and half the total number of attributes than *caulkins*. If we compare *caulkins* with *hughes-r* having both the same number of attributes, we can see how the processing of the *hughes-r* dataset requires a fraction of the time for processing *caulkins*. It is worth noticing that even when *hughes-r* has a quarter of the

number of *caulkins*'s objects (401 and 1685, respectively), this do not explains the difference in computation time and number of formal concepts. In fact, the clarified formal contexts for both datasets have similar sizes (1146×12 for *caulkins* and 1054×12 for *hughes-r*). Furthermore, it cannot even be explained in terms of the density of the formal context for which we have a small difference as shown in Table 2.

The only factor which explain this difference is established in the number of numerical attributes, which for *caulkins* are 9 out of 12 or three times those of *hughes-r* with 3 out of 12 numerical attributes. In general, a numerical attribute with a given θ-value generates a partition with more components and less elements per component than a categorical attribute, which yields a partition with less larger components. More components per pattern intent increment quadratically the number of intersections required per closure computation. This is confirmed by the statistics provided in Table 3 which shows that pattern intents in *hughes-r* have in average smaller components than *caulkins*. This phenomenon is not as easily graspable from the formal contexts nor the clarified version, however it can be understood as follows. For a large enough θ-value for a given attribute $m \in M$, every pair of objects will be similar for that attribute and thus, every pair of objects will have that attribute in the formal context. Consequently, the attribute m will subsume any other attribute in M, meaning that the closure of m with any other attribute is the top concept of the lattice. Differently, for a small enough θ-value, most pairs of objects will be not similar w.r.t. a given attribute. Actually, if no pairs of attributes are similar for m, the closure of $\{m\} \cup \{n\}$ for any $n \in M$, will be the bottom concept of the lattice (if we do not consider pairs (g_i, g_j) with $g_i = g_j$). Hence, it is for middle values of θ that we have the maximum number of possible concepts in the lattice. It is worth noticing that, while this is also true for categorical attributes, i.e. a unique category is equal to a large enough θ-value and a single category per object is similar to a low enough θ-value, the difference remains in the difference of the search spaces. Categorical attributes yield partitions, while numerical attributes yield tolerance blocks (with overlaps). The later, has a far larger search space than the first.

6.2.3. Synthesis

Finally, under the evidence shown by the experimental results, we can conclude that the use of pattern structures is of critical importance for mining similarity 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 far better option. We have also shown how setting the values of θ can greatly influence the output of our algorithm. The scripts and binaries necessary to replicate the experiments are freely available on-line[9].

[9] `http://liris.cnrs.fr/mehdi.kaytoue/alg/dam.experiments.tbz`

Dataset	θ-values
Diagnosis	$\theta_1 = 0.3, \theta_{2,3,4,5,6,7,8} = 0$
Contraceptive	$\theta_1 = 5, \theta_{2,3,4,5,6,7,8,9} = 0$
Servo	$\theta_{1,2,4,5} = 0, \theta_3 = 5$
Caulkins	$\theta_1 = 5, \theta_{2,3,4} = 0, \theta_{5,12} = 300, \theta_{6,7,10} = 2000, \theta_{8,9,11} = 10$
Hughes-r	$\theta_{1,2,3}, \theta_{4,5,6,7,8,9,10,11,12} = 0$
Pglw00	$\theta_1 = 5, \theta_{2,3,4,5,6} = 0$

Table 1: Theta values for each dataset. The corresponding subindex of the θ-value correspond to the attribute it was applied on. Comma separated values indicate more than one attribute.

Pattern structures $(M, (D, \sqcap), \delta)$						
Dataset	$\|M\|$	$\|T\|$=$\|G\|$	#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	$\|G\|$	$\|M\|$	#Con.	Num.	Cat.	Den. [%]	Exec. Time [s]
Diagnosis	7082	8	98	1	7	48.5	0.32 + 0.09
Contraceptive	1082307	10	1024	1	9	44.8	120.46 + 47.6
Servo	13688	5	28	1	4	38.1	0.54 + 0.1
Caulkins	1412827	12	2704	9	3	43.4	168.19 + 102.249
Hughes-r	80200	12	754	3	9	50.4	5.11 + 3.85
Pglw00	161892017	6	64	1	5	-	-

Formal context $(\mathfrak{B}_2(G), M, I)$ **clarified**							
Dataset	$\|G\|$	$\|M\|$	#Con.	Num.	Cat.	Den. [%]	Exec. Time [s]
Diagnosis	50	8	98	1	7	48.5	0.32 + 0.02
Contraceptive	1017	10	1024	1	9	50.0	120.46 + 0.089
Servo	25	5	28	1	4	48.4	0.54 + 0.006
Caulkins	1146	12	2704	9	3	47.4	168.19 + 0.169
Hughes-r	1054	12	754	3	9	49.5	5.11 + 0.063
Pglw00	-	6	64	1	5	-	-

Table 2: Datasets and execution times (Con. : Concepts. Num. : Numerical attributes. Cat. : Categorical attributes. Den. : Density ($\frac{|I|}{|G|\times|M|}$)). The symbol "-" indicates that the value could not be obtained by computational limitations.

Pattern structures				
Dataset	Mean elements	STD elements	Mean components	STD components
Diagnosis	19.63	15.72	25.72	25.01
Contraceptive	24.09	60.51	416.77	298.65
Servo	20.12	24.43	33.5	40.19
Caulkins	12.69	43.39	551.66	173.99
Hughes-r	29.36	31.66	25.46	20.49
Pglw00	1616.13	2008.25	88.3	139.16

Table 3: Statistics over the tolerance blocks in the pattern intents.

7. Related Work

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

Indeed, functional dependencies in their basic definition are not suitable for several tasks and data settings. As such a myriad of generalizations have been proposed over the last twenty years which were recently and exhaustively reviewed by Caruccio et. al [5]. Their classification separates (i) *extent relaxations* where FDs are relaxed on their coverage or conditions (e.g. the FD holds in a subset of data) from (ii) *attribute-comparison relaxations* where the notion of agreement between two objects for a given attribute is revisited. This includes (a) approximate-matching dependencies where the equality between two values is relaxed, and (b) ordered dependencies where an ordering between the values must be respected. This classification can be found in page 6 of [5].

Moreover, through the use of *symmetric relations* we can have a direct characterization of these approximate-matching dependencies in the framework of Formal Concept Analysis (FCA), both with data transformation and pattern structure techniques. The characterization in FCA consists in the creation of a formal context (G, M, I) such that G is formed by combining tuples of the original table, and M is formed also by combining the original attribute set of the table. Implications of such a formal context correspond exactly the set of functional dependencies [10, 32, 33, 34]. Thanks to *partition pattern structures*, this characterization can be achieved without an explicit transformation of the original data, making the computation of dependencies feasible [12].

Going back to the classification of [5], it can be noticed that the other type of dependencies that rely on attribute comparisons are those considering an *ordering criteria* (see (ii).(b) above). Such dependencies cannot be handled directly in our framework, as not only blocks of tolerances should be considered, but the ordering of the objects in each block should coincide as well [35, 5]. This problem is actually very close to closed gradual itemset mining [36] for which a first FCA characterization was given with an implicit pattern structure of sequential patterns [37].

8. Conclusion

In this paper, we present a generalization of functional dependencies, namely approximate-matching dependencies, which can take into account "fuzzy dependencies" in the sense that two attributes values are considered as "equal" as soon as they belong to a given interval, i.e. small variations of the attribute values are allowed. We discuss 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. We show that we can characterize this family using the formalism of pattern structures as this is also the case with standard

functional dependencies. 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 *symmetric relations* for comparing values between tuple attributes, just as *equivalence relations* (or equivalently, partitions) gives the underlying model for standard functional dependencies.

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

[5] 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.

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

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

[8] 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.

[9] 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.

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] G. Graetzer, B. Davey, R. Freese, B. Ganter, M. Greferath, P. Jipsen, H. Priestley, H. Rose, E. Schmidt, S. Schmidt, F. Wehrung, R. Wille, General Lattice Theory, Freeman, San Francisco, CA, 1971.

[18] 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.

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

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

[21] 12th International Workshop on the Web and Databases, WebDB 2009, Providence, Rhode Island, USA, June 28, 2009.

[22] U. of Dayton, Environmental Protection Agency Average Daily Temperature Archive, academic.udayton.edu/kissock/http/Weather/.

[23] 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.

[24] 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, Moscow, Russia, July 18-22, 2016., Vol. 1624 of CEUR Workshop Proceedings, CEUR-WS.org, 2016, pp. 123–134.

[25] A. Melton, S. Shenoi, Fuzzy relations and fuzzy relational databases, Computers and Mathematics with Applications 21 (11) (1991) 129 – 138.

[26] R. Belohlávek, V. Vychodil, Data Dependencies in Codd's Relational Model with Similarities, in: Handbook of Research on Fuzzy Information Processing in Databases, IGI Global, 2008, pp. 634–657.

[27] 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.

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

[29] S. O. Kuznetsov, Pattern Structures for Analyzing Complex Data, in: H. Sakai, M. K. Chakraborty, A. E. Hassanien, D. Slezak, W. Zhu (Eds.), RSFDGrC, Vol. 5908 of Lecture Notes in Computer Science, Springer, 2009, pp. 33–44.

[30] M. Kaytoue, S. O. Kuznetsov, A. Napoli, S. Duplessis, Mining gene expression data with pattern structures in Formal Concept Analysis, Information Sciences 181 (10) (2011) 1989–2001.

[31] 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.

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

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

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

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

[36] 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.

[37] 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.

LaTeX Source Files

Click here to download LaTeX Source Files: R2_submission.zip