Computing Similarity Dependencies with Pattern Structures

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

[1] Universitat Politècnica de Catalunya. 08032, Barcelona. Catalonia.
[2] Université de Lyon. CNRS, INSA-Lyon, LIRIS. UMR5205, F-69621, France.
[3] LORIA (CNRS - Inria Nancy Grand Est - Université de Lorraine), B.P. 239, F-54506, Vandœuvre-lès-Nancy.
jbaixer@lsi.upc.edu,mehdi.kaytoue@insa-lyon.fr,amedeo.napoli@loria.fr

Abstract. Functional dependencies provide valuable knowledge on the relations between the attributes of a data table. To extend their use, generalizations have been proposed, among which purity and approximate dependencies. After discussing those generalizations, we provide an alternative definition, the similarity dependencies, to handle a similarity relation between data-values, hence un-crisping the basic definition of functional dependencies. This work is rooted in formal concept analysis, and we show that similarity dependencies can be easily characterized and computed with pattern structures.

1 Introduction

In the relational database model, functional dependencies (FDs) are among the most popular types of dependencies [19] since they indicate a functional relation between sets of attributes: the values of a set of attributes are determined by the values of another set of attributes. To handle errors and uncertainty in real-world data, alternatives exist. *Approximate Dependencies* [12] are FDs that hold in a part –which is user defined– of the database. *Purity Dependencies* [15] express the relationship on the relative impurity induced by two partitions of the table (generated by two sets of attributes). If the impurity is zero, we have a FD.

These generalizations do not necessarily capture the semantics of some patterns that may hold in a dataset. This motivates the definition of "Similarity Dependencies", which can be seen as a generalization of Functional Dependencies, but un-crispring the basic definition of FDs: similar values of an attribute determine similar values of another attribute. Similarity has been considered for FDs under several terms, e.g. fuzzy FDs [3], matching dependencies [16], constraint generating dependencies [2]. Moreover, it is still an active topic of research in the database community [4,8,16,17].

The main objective of the present article is to give a characterization of similarity dependencies within FCA [10], thanks to the formalism of pattern structures [9]. Indeed, characterizing and computing FDs is strongly related to lattice theory and FCA. For example, the lattice characterization of a set of FDs is studied in [5,6,7], while a characterization within a formal context in

FCA is proposed in [10]. The latter is based on a *binarization*, which is the transformation of the original set of data into a binary context. To overcome the burden usually induced by such a transformation, pattern structures [9] have emerged as a valuable alternative to avoid arbitrary transformations and complexity problems [1].

Accordingly, our purpose here is threefold. Firstly, we propose a definition of *Similarity Dependencies*, and secondly a formalization based on pattern structures in FCA, avoiding a transformation of data into a binary table. It follows that classical algorithms of FCA can be –almost directly– applied to compute similarity dependencies. This work is based on [1] where FDs are characterized thanks to pattern structures, and on [13] where similarity is introduced in pattern structures as a tolerance relation (reflexive, symmetric, but not transitive). Finally, we also report preliminary experiments showing the capabilities of the approach.

The paper is organized as follows. In Section 2 we introduce the definition of Functional, Approximate and Purity Dependencies. In Section 3 we propose a definition and a characterization of Similarity Dependencies with pattern structures. Finally, Section 4 reports preliminary experimental results showing the capabilities of our approach.

2 Functional, Approximate and Purity 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.

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:

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

In Example 1, we have $t_2(\{a, c\}) = \langle t_2(a), t_2(c)\rangle = \langle 4, 4\rangle$. In this paper, the set notation is usually omitted and we write ab instead of $\{a, b\}$.

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	1	3	4	1
t_2	4	3	4	3
t_3	1	8	4	1
t_4	4	3	7	3

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. The set of partitions of a set is a lattice [11]. We recall that partitions can also be considered as equivalence classes induced by an equivalence relation.

Now, we define the set of the "maximal subsets" of a set.

Definition 1. *Given a finite base set S and $X = \{X_1, X_2, \ldots, X_n\}$ a set of subsets of S, a subset X_i is maximal in X if there does not exist any other subset X_j in X such that $X_i \subset X_j$.*

Then X_{Max} is the set of the maximal subsets of X.

For example, let $S = \{a, b, c\}$ and $X = \{\{a, b\}, \{b, c\}, \{a\}, \{b\}\}$. Then X is a subset of $\wp(S)$ the powerset of S, but not all elements of X are maximal subsets. Indeed, $X_{Max} = \{\{a, b\}, \{b, c\}\}$.

Moreover, we define the function max_S which applies to a set of sets such as X and returns the set of maximal subsets of X, i.e. X_{Max}.

Definition 2. *Given a finite set S and a subset $X = \{X_1, X_2, \ldots, X_n\}$ of $\wp(S)$, the function max_S returns the set X_{Max} of maximal subsets of X:*

$$max_S(X) = X_{Max} = \{X_i \in X \mid \nexists X_j \in X : X_i \subset X_j\}$$

2.2 Functional Dependencies

We now introduce functional dependencies (FDs).

Definition 3 ([19]). *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 Functional Dependencies 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 4. *Let $X \subseteq \mathcal{U}$ be a set of attributes in a table T. Two tuples t_i and t_j in T are equivalent w.r.t. X when:*

$$t_i \sim t_j \iff t_i(X) = t_j(X)$$

Then, the partition of T induced by X is a set of equivalence classes:

$$\Pi_X(T) = \{c_1, c_2, \ldots, c_m\}$$

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 or alternatively 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 Part(T). We can also notice that the set of partitions of any set Part(T) induces an ordering relation $\leq$:

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

For example: $\{\{t_1\}, \{t_2\}, \{t_3, t_4\}\} \leq \{\{t_1\}, \{t_2, t_3, t_4\}\}$. 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 ([12]). *A functional dependency $X \to Y$ holds in T if and only if $\Pi_Y(T) \leq \Pi_X(T)$.*

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

2.3 Purity and Approximate Dependencies

Approximate Dependencies [12]. In a table, there may be some tuples that prevent a functional dependency from holding. Those tuples can be seen as exceptions (or errors) for that dependency. Removing such tuples allows the dependency to exist: then a threshold can be set to define a set of "approximate dependencies" holding in a table. For example, a threshold of 10% means that all functional dependencies 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 functional dependency does not need to be the same for each approximate dependency. Considering in Example 2 the dependency $Month \to Av.Temp$, 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%, $Month \to Av.Temp$ is a valid Approximate Dependency.

Example 2. This table is an excerpt of the *Average Daily Temperature Archive* [4] from The University of Dayton, that shows the month average temperatures for different cities.

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

Purity Dependencies [15] are a generalization of the relationship between partitions induced by the left-hand side and right-hand side of a functional dependency. These dependencies are based on the relative impurity measure of two

partitions. In order to compute this impurity measure, we need a concave and subadditive function defined on the interval $[0, 1]$ (for example, the binary entropy function). The intuition about this measure is that it computes *how much those partitions disagree*, i.e. how far two partitions π and σ are from fulfilling the relation $\pi \leq \sigma$. If the impurity measure is zero (or close to zero), then $\pi \leq \sigma$.

For example, the impurity measure (details on this measure are given in [14]) of partition $\{\{1, 2, 3\}, \{4, 5\}\}$ w.r.t. partition $\{\{1, 2\}, \{3, 4, 5\}\}$ is 5.6, whereas the impurity measure of partition $\{\{1, 3\}, \{2, 5\}, \{4\}\}$ w.r.t. partition $\{\{1, 2\}, \{3, 4, 5\}\}$ is 8.2. In the first pair of partitions, only tuple 3 is misplaced, i.e. moving 3 from one partition to another leads to the the same partitions, whereas in the second example, the number of misplaced elements is larger (2, 3, and 4 should be moved).

An important feature of this measure is that if a partition is finer than another, then, their relative impurity measure is exactly 0. This implies that a purity dependency $X \to Y$ holds if and only if the relative impurity of $\Pi_X(T)$ w.r.t. $\Pi_Y(T)$ is below a user-defined threshold. Therefore, if $\Pi_Y(T) \leq \Pi_X(T)$, a functional dependency is a valid purity dependency, regardless of the threshold.

For example, we consider all the possible dependencies having the attribute *Average Temperature* in their right-hand side. The purpose of this choice is to find out which attributes determine the values of the average temperature (Av. Temp.) in Example 2. Considering Approximate Dependencies, we introduce the two metrics *# Tuples* and *Percentage*: *# Tuples* denotes the minimal number of tuples that must be removed from the dataset for allowing the dependency to hold, and *Percentage* denotes the percentage that *# Tuples* represents for the whole dataset. For example, the Approximate Dependency $Month \to Av.Temp$ holds when we remove at least 6 (well-chosen) tuples, which represent 75% of the whole dataset.

Example 3. Dependencies with *Average Temperature* in their right-hand and the metrics related to Approximate and Purity Dependencies.

Dependency	#Tuples	Percentage	Purity
`Month -> Av. Temp`	6	75%	12.98
`Month, Year -> Av. Temp`	1	12.5%	4.0
`Month, City -> Av. Temp`	4	50%	4.0
`Year -> Av. Temp`	3	37.5%	8.26
`Year, City -> Av. Temp`	0	0%	0.0
`City -> Av. Temp`	4	50%	4.0

As for the purity measure, we use the measure of relative entropy of two partitions described in [14]. If we examine the dependency $Month \to Av.Temp$, we should the relative entropy of the partitions induced by *Month* and *Av. Temp.*, which are, respectively:

$\Pi_{Month} = \{\{t_1, t_2, t_5, t_6\}, \{t_3, t_4, t_7, t_8\}\}$

$\Pi_{Av.Temp.} = \{\{t_1\}, \{t_2\}, \{t_3\}, \{t_4\}, \{t_5\}, \{t_6\}, \{t_7\}, \{t_8\}\}$

Then, the relative entropy of Π_{Month} and $\Pi_{Av.Temp.}$ is 12.98, i.e. the largest of the conditional entropies that are computed. Actually the number of tuples that need to be reallocated for $\Pi_{Av.Temp.} \leq \Pi_{Month}$ is significantly large. It is also significant that the number of tuples that need to be removed for the dependency $Year, City \rightarrow Av.Temp$ to hold is zero and that the relative entropy of $\Pi_{Year,City}$ and $\Pi_{Av.Temp.}$ is zero as well. The Functional Dependency $Year, City \rightarrow Av.Temp$ holds because there is no pair of tuples t_i, t_j such that $t_i(Year, City) = t_j(Av.Temp.)$, i.e. there is no need to remove any tuple to verify this dependency. In addition the relative entropy of $\Pi_{Year,City}$ and $\Pi_{Av.Temp.}$ is zero, because the partitions induced by both sides, $\Pi_{Year,City}$ and $\Pi_{Av.Temp}$ are exactly the same: $\{\{t_1\}, \{t_2\}, \{t_3\}, \{t_4\}, \{t_5\}, \{t_6\}, \{t_7\}, \{t_8\}\}$. Therefore, the relation $\Pi_{Year,City} \leq \Pi_{Av.Temp}$ holds, i.e. the relative entropy is zero and this dependency trivially holds.

Yet, the intuition about this dataset is that the "Average Temperature" depends, to some extent, on the location and the month, i.e. given a city and a month, we should be able to predict the average temperature. But this intuitive relationship is somehow difficult to deduce with Approximate and Purity Dependencies. For example, the metrics for the dependency $Month, City \rightarrow Av.Temp$ indicate that 4 tuples must be removed (50% of the dataset) for checking this dependency, or alternatively, the relative entropy of the partitions $\Pi_{Month,City}$ and $\Pi_{Av.Temp}$ is 4.0. Considering the number of tuples, removing 50% of the whole dataset is a lot, especially if the intuition tells that this dependency should hold. Considering the entropy rate, the smallest entropy rate is zero and the largest computed rate is 12.98. Thus, it seems difficult to deduce the right threshold in each case.

Instead of considering measures that deal with the sets of tuples as a whole, dependencies could be directly related with the notion of "similarity": if two tuples have similar values for the attributes *Month* and *City*, then they should have a similar value for the attribute *Av. Temp*. This can be interpreted as follows: if two cities are close enough and the corresponding months are also close enough, then the average temperature in the cities should be close enough or "similar" as well. In such a context, *"having similar values"* depends on the type of the attributes. For temperatures it mean that the absolute difference of the values is less than a given threshold. For months, it could mean that they are adjacent. For cities, it could mean that their locations are close enough.

Such a kind of dependency would provide more control and a more intuitive explanation of the relations existing between attributes.

3 Similarity Dependencies

First, we define a *tolerance* relation in a set S:

Definition 5. *$\theta \subseteq S \times S$ is a tolerance relation if:*

1. $\forall s_i \in S : s_i \theta s_i$ *(reflexivity)*
2. $\forall s_i, s_j \in S : s_i \theta s_j \iff s_j \theta s_i$ *(symmetry)*

A tolerance relation is not necessarily transitive and induces *blocks of tolerance*:

Definition 6. *Given a set S, a subset $K \subseteq S$, and a tolerance 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)*

All elements in a tolerance block are in pairwise correspondence, and the block is maximal with respect to the relation θ. The set of all tolerance blocks induced by a tolerance relation θ on the set S is denoted by S/θ (by analogy with the notation of equivalence classes). S/θ is a set of maximal subsets of S and as such, $S/\theta \in \wp(\wp(S))$. Thus we have:

Property 1. $\forall K_i, K_j \in S/\theta : K_i \not\subseteq K_j$ and $K_j \not\subseteq K_i$ for all $i \neq j$

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

Definition 7. *Let θ_1 and θ_2 two tolerance 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 tolerance relations θ_1 and θ_2, or, equivalently, on the sets of blocks of tolerance of θ_1 and θ_2 are:

Definition 8. *Let θ_1 and θ_2 two tolerance relations in the set S.*
$\theta_1 \wedge \theta_2 = \theta_1 \cap \theta_2 = \max_S(\{k_i \cap k_j \mid k_i \in S/\theta_1, k_j \in S/\theta_2\})$
$\theta_1 \vee \theta_2 = \theta_1 \cup \theta_2 = \max_S(S/\theta_1 \cup S/\theta_2)$

The meet $\theta_1 \wedge \theta_2$ is the set of pairwise intersections of all blocks in S/θ_1 and S/θ_2, and then removing intersections that are not maximal. The join is simpler as it consists in simply joining the blocks of tolerance in S/θ_1 and S/θ_2 and then removing the unions that are not maximal.

An example of a tolerance 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 transitivity does not apply.

We now come back to the set of tuples T and the set of attributes M. For each attribute $m \in M$, we define a tolerance relation on the values of that attribute: θ_m. The set of tolerance blocks induced by the tolerance relation of the attribute m is 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. the attribute m.

Example 4. Let us define a tolerance relation on an attribute $m \in \{a, b, c, d\}$ as follows: $t_i\theta_m t_j \iff |t_i(m) - t_j(m)| \leq \epsilon$.

Now, assuming that $\epsilon = 1$ in example 1, we have:

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

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* attributes in X. We now can define a *similarity dependency*:

Definition 9. *Let $X, Y \subseteq \mathcal{U}$: $X \to Y$ is a similarity dependency iff: $\forall t_i, t_j \in T : t_i \theta_X t_j \Rightarrow t_i \theta_Y t_j$*

In the case of a functional dependency, $X \to Y$ holds if and only if, for each pair of tuples having the *same value* w.r.t. the attributes in X, then, they have the *same value* w.r.t. the attributes in Y.

In the case of a similarity dependency, $X \to Y$ holds if and only if, for each pair of tuples having *similar values* w.r.t. the attributes in X, then, they have *similar values* w.r.t. the attributes in Y.

Example 5. We revisit the table in Example 4 and we define the tolerance relation: $t_i \theta_m t_j \iff |t_i(m) - t_j(m)| \leq 2$. Then the following similarity 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 a similarity 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 similarity dependency $bcd \to a$ does not hold, as $t_1 \theta_{bcd} t_2$ 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 6. Going back to example 2, let us compute the Similarity Dependencies that hold and that have the attribute *Av. Temp.* in their right-hand side).

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 similarity dependency that holds is $Month, City \to Av.Temp$, using the following similarity measures for each attribute: $x \ \theta_{Month} \ y \iff |x - y| \leq 0$, $x \ \theta_{Year} \ y \iff |x - y| \leq 0$, $x \ \theta_{City} \ y \iff distance(x, y) \leq 500$ and $x \ \theta_{Av.Temp} \ y \iff |x - y| \leq 10$.

The similarity imposes that the month and year must be the same, whereas the distance between cities should be less than 500 kilometers and the difference between average temperatures should be less than 10 degrees (all these values are of course 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 $|36.4 - 33.8| \leq 10$.

3.1 Computing Similarity Dependencies with Pattern Structures

A pattern structure allows one to apply FCA directly on non-binary data [9]. Formally, let G be a set of objects, let $(D, \sqcap)$ be a meet-semi-lattice of potential object descriptions and let $\delta : G \longrightarrow D$ 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 $\sqsubseteq$: $\forall c, d \in D$, $c \sqsubseteq d \iff c \sqcap d = c$. A pattern structure $(G, (D, \sqcap), \delta)$ is based on two derivation operators $(\cdot)^{\square}$. 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.

Thanks to the formalism of pattern structures, similarity dependencies can be characterized (and computed) in an elegant manner. 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 tolerance relations can be ordered as presented and discussed in Definitions 7 and 8, then descriptions can be ordered within a lattice.

Then, 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 defined in Definition 8.

An example of concept formation is given as follows. 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:

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

This pattern concept lattice allows us to characterize all similarity dependencies holding in M:

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

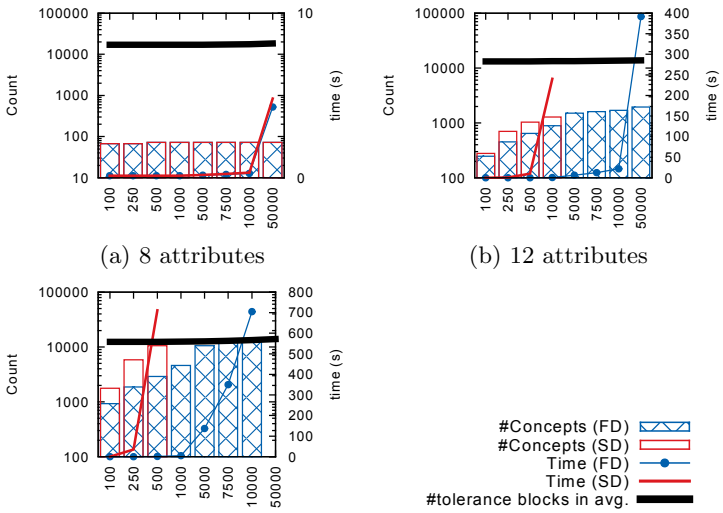
Proof. First of all, we notice that $(t, t') \in \{X\}^{\square}$ if and only if $t(X)\theta_X t'(X)$, since $(t, t') \in \{X\}^{\square}$ if and only if $\forall x \in X : t(x)\theta_x t'(x)$, if and only if $t(X)\theta_X t'(X)$. We also notice that $\{X, Y\}^{\square} \subseteq \{X\}^{\square}$.
($\Rightarrow$) We prove that if $X \to Y$ holds in T, then, $\{X\}^{\square} = \{X, Y\}^{\square}$, that is, $\{X\}^{\square} \subseteq \{X, Y\}^{\square}$. We take an arbitrary pair $(t, t') \in \{X\}^{\square}$, that is: $t(X)\theta_X t'(X)$.

Since $X \rightarrow Y$ 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, $(t,t') \in \{XY\}^{\square}$, that is: $t(XY)\theta_{XY}t'(XY)$. Since this is for all pairs $t,t' \in T$ such that $t(X) = t'(X)$, this implies that $X \rightarrow Y$ holds in T.

4 Experiments

Dataset description. Electronic sport denotes the extreme practice of video games where so-called cyber-athletes compete in world-wide tournaments. As for any sport, such professionals are surrounded by sponsors and practice within professional teams. These professional games are even broadcast by commentators over specialized TV channels [18]. StarCraft II (Blizzard Entertainment) is the most competitive video game. Since e-sport is a digital entertainment, one can easily find game statistics and recording in great numbers on the Web. We list more than $209,000$ games between two opponents and their associated statistics (attributes). For each game, we derive two tuples (one for each of the players involved). Each player in a game (tuple) is described by 31 attributes such as his faction, the result, and several indicators of his game play.
Experimental settings. The final dataset has about $400,000$ tuples described by 31 attributes with different domain types (Boolean, qualitative, and numerical). For attributes with Boolean or non-ordered qualitative domains, the similarity parameters are set to 0 as for classical FDs, since we do not have similarity constraints between their values. For the others, parameters are set by an expert of the domain, helped with the distribution of that attribute values. Thanks to the genericity of pattern structures, we slightly modified the very simplistic Java version of CloseByOne used in [1] for extracting classical FDs. The only modification lies in building the descriptions, i.e. producing tolerance blocks instead of partitions over the set of objects. We experimented on 1.8GHz and 4GB of RAM machines.
Preliminary results. We build first several different sub-datasets with randomly chosen set of tuples and different subsets of attributes. We report execution times for extracting pattern concepts (and their count) to characterize functional dependencies and similarity dependencies for three datasets in Figure 1. We also report the average number of tolerance classes of each attribute (that allows to build their description). In Figure 1 (a), the dataset is composed of qualitative and not comparable attributes only. Thus, we set the similarity parameters to 0 and observe that extracting FDs and SDs requires the same amount of time and returns the same concepts (since setting the similarity parameter to 0 leads to partitions). Naturally, the number of tolerance blocks is high, corresponding to the cardinality of attribute domains. We added more attributes, among which 5 numerical, and introduce the similarity parameters, see Figure 1 (b) and (c). The number of tolerance blocks is still high since it is an average for all attributes, and attribute domains are very dense. We notice that

there are more concepts to characterize SDs than FDs. This is due to our choice of similarity parameters. Finally, we face memory issues when computing pattern concepts for SDs, when the algorithms terminates for FDs. This is due to our pattern implementations, i.e. how a pattern is represented in the memory. We used *striped partitions*, i.e. not store any part of size 1, which can strongly reduce the pattern size in memory. For FDs, this experimentally happens more often than for SDs, due to the relaxation of the equality constraint. We need to investigate other pattern implementations.



(a) 8 attributes (b) 12 attributes

(c) 17 attributes (d) Legend

Fig. 1. Experimental results (X-axis represents the number of objects/tuples)

5 Conclusion

We discussed how Functional, Approximate and Purity Dependencies may not capture some relationships among attributes that intuitively exist in a dataset. We presented alternatively Similarity Dependencies, to express relationships between attributes based on a similarity measure that depends on the semantics of each attribute. We showed that similarity dependencies are easily characterized in FCA with pattern structures and we gave a preliminary experimental study.

Future work is in concern with a deeper investigation of the best in-memory pattern representations for fast and scalable computation, the introduction of a minimal support as well as a qualitative evaluation of similarity dependencies.

Acknowledgments. This research work has been supported by the Spanish Ministry of Education and Science (project TIN2008-06582-C03-01), EU PASCAL2 Network of Excellence, and by the Generalitat de Catalunya (2009-SGR-980 and 2009-SGR-1428) and AGAUR (grant 2010PIV00057).

References

1. J. Baixeries, M. Kaytoue, and A. Napoli. Computing functional dependencies with pattern structures. In L. Szathmary and U. Priss, editors, *CLA*, volume 972 of *CEUR Workshop Proceedings*, pages 175–186. CEUR-WS.org, 2012.
2. M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999.
3. R. Belohlávek and V. Vychodil. Data tables with similarity relations: Functional dependencies, complete rules and non-redundant bases. In M.-L. Lee, K.-L. Tan, and V. Wuwongse, editors, *DASFAA*, volume 3882 of *Lecture Notes in Computer Science*, pages 644–658. Springer, 2006.
4. L. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *Proceedings of the 14th International Conference on Database Theory*, ICDT '11, pages 268–279, New York, NY, USA, 2011. ACM.
5. N. Caspard and B. Monjardet. The lattices of closure systems, closure operators, and implicational systems on a finite set: A survey. *Discrete Applied Mathematics*, 127(2):241–269, 2003.
6. A. Day. The lattice theory of fonctionnal dependencies and normal decompositions. *International Journal of Algebra and Computation*, 02(04):409–431, 1992.
7. J. Demetrovics, G. Hencsey, L. Libkin, and I. B. Muchnik. Normal form relation schemes: A new characterization. *Acta Cybern.*, 10(3):141–153, 1992.
8. W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *The VLDB Journal*, 20(4):495–520, Aug. 2011.
9. B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In H. S. Delugach and G. Stumme, editors, *Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001)*, LNCS 2120, pages 129–142. Springer, 2001.
10. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, Berlin, 1999.
11. G. Graetzer, B. Davey, R. Freese, B. Ganter, M. Greferath, P. Jipsen, H. Priestley, H. Rose, E. Schmidt, S. Schmidt, F. Wehrung, and R. Wille. *General Lattice Theory*. Freeman, San Francisco, CA, 1971.
12. Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal*, 42(2):100–111, 1999.
13. M. Kaytoue, Z. Assaghir, A. Napoli, and S. O. Kuznetsov. Embedding tolerance relations in formal concept analysis: an application in information fusion. In J. Huang, N. Koudas, G. J. F. Jones, X. Wu, K. Collins-Thompson, and A. An, editors, *CIKM*, pages 1689–1692. ACM, 2010.
14. D. Simovici and S. Jaroszewicz. An axiomatization of partition entropy. *Information Theory, IEEE Transactions on*, 48(7):2138–2142, 2002.
15. D. A. Simovici, D. Cristofor, and L. Cristofor. Impurity measures in databases. *Acta Inf.*, 38(5):307–324, 2002.
16. S. Song and L. Chen. Efficient discovery of similarity constraints for matching dependencies. *Data & Knowledge Engineering*, (0):–, 2013. (in press).
17. S. Song, L. Chen, and P. S. Yu. Comparable dependencies over heterogeneous data. *The VLDB Journal*, 22(2):253–274, Apr. 2013.
18. T. L. Taylor. *Raising the Stakes: E-Sports and the Professionalization of Computer Gaming*. MIT Press, 2012.
19. J. Ullman. *Principles of Database Systems and Knowledge-Based Systems, volumes 1–2*. Computer Science Press, Rockville (MD), USA, 1989.