Computing Functional Dependencies with Pattern Structures

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

[1] Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya. 08032 Barcelona. Catalonia.
jbaixer@lsi.upc.edu
[2] Université de Lyon. CNRS, INSA-Lyon, LIRIS. UMR5205, F-69621, France.
mehdi.kaytoue@insa-lyon.fr
[3] LORIA, B.P. 70239, Équipe Orpailleur, Bâtiment B, F-54506 Vandœuvre-lès-Nancy
amedeo.napoli@loria.fr

Abstract. The treatment of many-valued data with FCA has been achieved by means of scaling. This method has some drawbacks, since the size of the resulting formal contexts depends usually on the number of different values that are present in a table, which can be very large.
Pattern structures have been proved to deal with many-valued data, offering a viable and sound alternative to scaling in order to represent and analyze sets of many-valued data with FCA.
Functional dependencies have already been dealt with FCA using the binarization of a table, that is, creating a formal context out of a set of data. Unfortunately, although this method is standard and simple, it has an important drawback, which is the fact that the resulting context is quadratic in number of objects w.r.t. the original set of data.
In this paper, we examine how we can extract the functional dependencies that hold in a set of data using pattern structures. This allows to build an equivalent concept lattice avoiding the step of binarization, and thus comes with better concept representation and computation.

Keywords: Association rules and data dependencies, attribute implications, data dependencies, pattern structures, formal concept analysis

1 Introduction and Motivation

In the relational database model there are different types of dependencies ([1,18]). Functional dependencies are among the most popular. The reason is that they are important in order to explain the normalization of a database scheme in the Relational Database Model. Functional Dependencies (FD's) have their own set of axioms ([5,18]), which, in turn, are also shared by other dependencies. For instance, implications share the same axioms as functional dependencies ([2]), which are basically reflexivity, augmentation and transitivity.

These axioms state how functional dependencies behave in the presence of a set of dependencies of the same kind. For instance, we can decide whether a set of functional dependencies Σ implies a single FD σ, that is, $\Sigma \models \sigma$

ISBN 978–84–695–5252–0, Universidad de Málaga (Dept. Matemática Aplicada)

We can also find a minimal set of functional dependencies that implies a given set of them, that is, we can compute Σ' such that $\Sigma' \models \Sigma$, where Σ' is minimal. In this case, we also say that Σ' is the minimal generating set of Σ. These two problems have been studied in [1] and [16], and algorithms have been proposed. Yet, it is important to note that this calculation is performed starting from a set of dependencies, not a set of data.

In this paper, we aim at finding a characterization of functional dependencies that hold in a set of data using Formal Concept Analysis and pattern structures.

The lattice characterization of a set of Functional Dependencies is studied in [6,7,8,9], and the characterization with a formal context in [3,12]. This characterization is based on a *binarization*, which is the transformation of the original set of data into a binary context.

In fact, the primary concern when computing the characterization of a set of functional dependencies with FCA is that, generally, the dataset is many-valued, and not binary. This means that the set of data must be somehow transformed to obtain a binary context.

Applying conceptual scaling (without information loss) results either in a larger set of objects in the resulting formal context, or a larger set of attributes.

On another hand, pattern structures ([11,14]) have emerged as a valid alternative to work with non binary contexts and specially with numerical contexts, as well as to avoid the complexity drawbacks that are present in scaling.

Therefore, we have two different methods of computing the characterization of a set of functional dependencies that hold in a set of data:

1. Binarizing or scaling the original set of data, and obtaining a formal context.
2. Using pattern structures.

The purpose of this paper is twofold. On the one hand, we propose using pattern structures as a way to compute the characterization of a set of functional dependencies that hold in a set of data. The interest is to prove that pattern structures is a flexible mechanism that may encode the semantics of functional dependencies without adding further penalty to the resulting formal context. On the other hand, we aim at setting up a solid connection between the formalism of pattern structures and the finding of other different kinds of dependencies that may hold in a given set of data.

The paper is organized as follows. Firstly, the definitions of functional dependencies and their axioms are explained in Section 2 together with the scaling procedure allowing one to derive a formal context from a numerical dataset that characterize FD. Then, Section 3 presents the general formalism of pattern structures. Section 4 gives the core of this article: it shows how to define a pattern structure that holds the same concept lattice than with the introduced scaling. It follows experiments in Section 5 showing the interest of using pattern structures. Finally, the conclusion draws attention to perspectives of research.

2 Motivating Example

This example shows how functional dependencies can be extracted using FCA (this is based on [4]). The main idea behind this method is called *binarization*, and consists in transforming (implicitly) a many-valued set of data into a binary context. This transformation allows us to build a formal context.

Before explaining this process, we first introduce functional dependencies (FD's). Let $\mathcal{U}$ be a set of attributes, and let Dom be a set of values (a domain). For the sake of clarity, we assume that Dom is a numerical set. A tuple t is a function $t : \mathcal{U} \mapsto Dom$, and a table T is a set of tuples. Usually tables are presented as a matrix, as in the following example:

id	A	B	C	D
t_1	1	3	7	2
t_2	1	3	4	5
t_3	3	5	2	2
t_4	3	3	4	8

where the set of tuples (or objects) is $\{ t_1, t_2, t_3, t_4 \}$ and $\mathcal{U} = \{ A, B, C, D \}$ is the set of attributes.

Given a tuple $t \in T$, we say that $t(X)$ (for all $X \subseteq \mathcal{U}$) is the restriction of the tuple t in the attributes $X \subseteq \mathcal{U}$, this is the values of t in the attributes X. For instance, we have that $t_2(\{ A, C \})$ is $\{ 1, 4 \}$. We drop the set notation and say that $t_2(AC)$ is $\{ 1, 4 \}$.

Definition 1 ([18]). *Let T be a set of tuples, and $X, Y \subseteq \mathcal{U}$. A* **functional dependency (FD)** *$X \to Y$, holds in T if:*

$$\forall t_i, t_j \in T : t_i(X) = t_j(X) \Rightarrow t_i(Y) = t_j(Y)$$

For instance, we have that the functional dependency $C \to B$ holds in T, whereas the functional dependency $A \to B$ does not, because $t_3(A) = t_4(A)$ but $t_3(B) \neq t_4(B)$.

We are now ready to explain how to extract the set of functional dependencies that hold in a set of data, using FCA:

1. We define a formal context derived from the original many-valued data T.
2. We extract the implications that hold in the concept lattice associated to that context ([12]).
3. We see that the implications that hold in the concept lattice are the functional dependencies that hold in the original table T.

In order to define a formal context, we need to define first the set of objects:

$$G = \{ (t_i, t_j) \mid i < j \text{ and } t_i, t_j \in T \}$$

It corresponds to the set of all pairs of tuples from T (excluding symmetry and reflexivity). The relation of the context is defined as:

$$(t_i, t_j)\ I\ x \Leftrightarrow t_i(x) = t_j(x)$$

It is important to realize that the formal context $\mathbb{K} = (G, \mathcal{U}, I)$ depends entirely on the table T, since both G and $\mathcal{U}$ do, but this dependency is not explicitly shown in the definition of this context. According to the preceding case, we would have the formal context in Figure 1 and the corresponding concept lattice in Figure 2.

$\mathbb{K}$	A	B	C	D
(1,2)	×	×		
(1,3)				×
(1,4)		×		
(2,3)				
(2,4)		×	×	
(3,4)	×			

Fig. 1. Formal Context

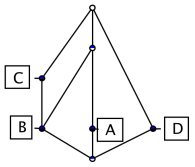

Fig. 2. Concept Lattice

We have created a new formal context out of a multi-valued table. We can see that the size of this context can be of the order of $\mathcal{O}(|T^2|)$ (where $|T|$ is the number of tuples of T), so it can be significantly bigger that the original set of data.

The following step is to compute the Duquenne-Guigues basis ([10]) of the concept lattice, i.e. the minimal set of implications such that all the implications that hold in the formal context can be derived from this set. In this case, this set consists of the following implications:

$$c \rightarrow b, abc \rightarrow d, bcd \rightarrow a$$

This is not the set of all implications that hold in the concept lattice of the formal context that we have defined. This is just a minimal set such that all the implications that hold in the lattice can be derived (by augmentation and transitivity) from this set. This means that, if Σ^* is the set of all implications that hold in the concept lattice, then $\Sigma \models \Sigma^*$.

Finally, we have to realize that the set of implications that hold in the concept lattice are syntactically the same as the set of functional dependencies that hold in T (this is shown in [4,12]). By *syntactically* we mean that whenever an implication $X \rightarrow Y$ holds in $\mathbb{K}$, then the functional dependency $X \rightarrow Y$ holds in T. Equivalently, the minimal generating set of functional dependencies that hold in T is the same as the Duquenne-Guigues basis of the concept lattice.

A known result of the binarization process precisely states that the set of implications that hold in the context is syntactically equivalent to the set of functional dependencies that hold in the original set of data [12].

3 Pattern Structures in Formal Concept Analysis

We assume the reader to be familiar with basic notions of formal concept analysis. We use the standard notations from [12]. Our interest lies in handling numerical data within FCA. Hence, we recall here the formalism of pattern structures that can be understood as a generalization towards complex data, i.e. objects taking descriptions in a partially ordered set.

A pattern structure is defined as a generalization of a formal context describing complex data [11]. 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 by a subsumption relation $\sqsubseteq$: $\forall c, d \in D$, $c \sqsubseteq d \Longleftrightarrow c \sqcap d = c$. A pattern structure $(G, (D, \sqcap), \delta)$ gives rise to two derivation operators $(\cdot)^{\square}$:

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

These operators form a Galois connection between $(2^G, \subseteq)$ 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. Existing FCA algorithms [15] can be used with slight modifications to compute pattern structures, in order to extract and classify concepts (details in [11,14]).

4 Finding FD with Partition Pattern Structures

As introduced in Section 2, an existing binarization allows to build a concept lattice from which a set of FDs can be characterized [12]. However, this formal context tends to be very large, even when the initial data are of reasonable size. We show here how the formalism of pattern structures can be instantiated to obtain an equivalent concept lattice. The so called *partition pattern structures* come with several advantages among which computation and interpretation of the resulting lattice.

4.1 Preliminaries on the partition lattice

Partition of a set. Given a set E, a partition over E is a set $P \subseteq \wp(E)$ s.t.:

- $\bigcup_{p_i \in P} p_i = E$
- $p_i \cap p_j = \emptyset$, for any $p_i, p_j \in P$ with $i \neq j$.

In other words, a partition covers E and is composed of disjoint subsets of E.

Equivalence relation. A partition P over a set E is an equivalence relation R_P on E. The 1-1-correspondence between P and R_P is given by $(e, e') \in R_P$ iff e and e' belongs to the same class of P [6,13]. For example, given $P = \{\{1, 2, 3\}, \{4\}\}$, one has the relation $R_P = \{(1, 2), (1, 3), (2, 3), (1, 1), (2, 2), (3, 3), (4, 4)\}$ (omitting symmetry for the sake of readability).

Ordering relation. A partition P_1 is finer than a partition P_2 (P_2 coarser than P_1), written $P_1 \sqsubseteq P_2$ if any subset of P_1 is a subset of a subset in P_2. For example,

$$\{\{1, 3\}, \{2\}, \{4\}\} \sqsubseteq \{\{1, 2, 3\}, \{4\}\}$$

Meet of two partitions. It is defined as the coarsest common refinement. In other words, it is the intersection of the respective equivalence relations:

$$\{\{1, 3\}, \{2, 4\}\} \sqcap \{\{1, 2, 3\}, \{4\}\} = \{\{1, 3\}, \{2\}, \{4\}\}$$
$$\textit{or } \{(1, 3), (2, 4)\} \cap \{(1, 2), (1, 3), (2, 3)\}$$

Join of two partitions. It is defined as the finest common coarsening. In other words, it is the transitive closure of the union of the respective equivalence relations.

$$\{\{1, 3\}, \{2\}\{, 4\}\} \sqcup \{\{1, 2\}, \{3\}\{4\}\} = \{\{1, 2, 3\}, \{4\}\}$$
$$\textit{or } \textit{transitive\_closure}(\{(1, 3), (4, 4)\} \cup \{(1, 2)(3, 3), (4, 4)\})$$

Finally, one should notice that the property $P_1 \sqcap P_2 = P_1 \Leftrightarrow P_1 \sqsubseteq P_2$ naturally holds (and the dual for join). Thus the set of all partitions over a set forms a lattice $(D, \sqcap, \sqcup)$ and can be used as a description space of a pattern structure.

4.2 Partition pattern structure

Consider a numerical table as a many-valued context (G, M, W, I) where G corresponds to the set of objects ("rows"), M to the set of attributes ("columns"), W the data domain ("all distinct values of the table") and $I \subseteq G \times M \times W$ a relation such that $(g, m, w) \in I$ also written $m(g) = w$ means that attribute m takes the value w for the object g [12]. In Table 1 (left), we have $D(4) = 8$.

We show how a partition pattern structure can be defined from a many-valued context (G, M, W, I) and show that its concept lattice is equivalent to the concept lattice obtained after binarization (see Section 2). Intuitively, formal objects of the pattern structure are the attributes of the numerical dataset. Then, given an attribute $m \in M$, its description $\delta(m)$ is given by a partition over G such that any two elements g, h of the same class take the same values for the attribute m, i.e. $m(g) = m(h)$. The result is given in Table 1 (middle). As such, descriptions

obey the ordering of a partition lattice as described above. It follows that our initial numerical table (G, M, W, I) can be represented as a pattern structure $(M, (D, \sqcap, \sqcup), \delta)$ where M is the set of original attributes, and $(D, \sqcap, \sqcup)$ is the lattice of partitions over G. An example of concept formation is given as follows, starting from set $\{A, B\} \subseteq M$:

$$\begin{aligned} \{A,B\}^{\square} &= \delta(A) \sqcap \delta(B) \\ &= \{\{1,2\},\{3,4\}\} \sqcap \{\{1,2,4\},\{3\}\} \\ &= \{\{1,2\},\{3\},\{4\}\} \\ \{\{1,2\},\{3\},\{4\}\}^{\square} &= \{m \in M | \{\{1,2\},\{3\},\{4\} \sqsubseteq \delta(m)\} \\ &= \{A,B\} \end{aligned}$$

Hence, $(\{A, B\}, \{\{1, 2\}, \{3\}, \{4\}\})$ is pattern concept. The resulting pattern concept lattice is given in Table 1 (left).

id	A	B	C	D
1	1	3	7	2
2	1	3	4	5
3	3	5	2	2
4	3	3	4	8

$m \in M$	$\delta(m) \in (D, \sqcap, \sqcup)$
A	$\{\{1,2\},\{3,4\}\}$
B	$\{\{1,2,4\},\{3\}\}$
C	$\{\{1\},\{2,4\},\{3\}\}$
D	$\{\{1,3\},\{2\},\{4\}\}$

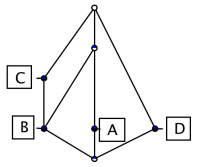

Table 1. The original data (left), the resulting pattern structure (middle) and its pattern concept lattice (right)

4.3 Formal context of the partition pattern structure

We showed that a numerical dataset can be described by a many-valued context (G, M, W, I) from which one can derive the formal context $(M, \mathcal{B}_2(G), I)$ [4] where $\mathcal{B}_2(G)$ represents any pair of objects, and $(m, (g, h)) \in I$ means that $m(g) = m(h)$. The resulting concept lattice can be used to extract FD [12]. A result is that both introduced structures $(M, \mathcal{B}_2(G), I)$ and $(M, (D, \sqcap), \delta)$ are equivalent, i.e. both collections of concepts are in 1-1-correspondence.

Proposition. Let (G, W, M, I) be a many-valued context. Let $(M, \mathcal{B}_2(G), I)$ be the formal context such as $(m, (g, h)) \in I \Leftrightarrow m(g) = m(h)$. Let $(M, (D, \sqcap, \sqcup), \delta)$ be the partition pattern structure where, for $m \in M$, $\delta(m)$ is the partition of G such that $p, q \in P$, $P \in \delta(m) \Leftrightarrow m(p) = m(q)$. Then, the following holds.
1. For any formal concept (R, S), there is one and only one pattern concept (C, d) such that $R = C$ and S is the equivalence class representation of d.
2. and vice-versa.

[4] Originally $(\mathcal{B}_2(G), M, I)$, but for sake of simplicity here, we permute formal objects and attributes. The results hold equally.

Proof. Consider both structures $(M, \mathcal{B}_2(G), I)$ and $(M, (D, \sqcap, \sqcup), \delta)$. They both hold the same set of "formal objects" M (attributes in the many-valued context). In the pattern structure, elements of M are described by a partition over G. In the formal context, elements are described by pairs of objects, that is, by definition, the representation of the same partitions. As such, elements of M are described in an equivalent way. Furthermore, intersections in both representations are equivalent too. Indeed, the meet operation between two partitions is known to be the intersection of their equivalence class representation. Since it is known that derivation operations are defined in pattern structures in the same way than in formal contexts, the proposition naturally holds.

Example. The pattern concept $(\{B\}, \{\{1,2,4\},\{3\}\})$ is equivalent to the formal concept $(\{B\}, \{(1,2),(2,4),(1,4)\})$.

From this example, one should remark that pattern structures offer more concise intent representation when the set of object becomes very large, i.e. storing a partition instead of all pairs of objects that are together in a same class of the partition. This leads us to the next section, where the attention is drawn to a computational comparison of both approaches.

5 Experiments

We showed how pattern structures can alternatively represent the formal context $(M, \mathcal{B}_2(G), I)$ by means of partition patterns. Both concept lattices are equivalent and thus can be used to characterize FD. To assess the usefulness of introducing partition pattern structures, we applied both methods to well known UCI datasets[5]. To compute with formal contexts, we wrote a simple procedure to scale the many-valued context into a formal context, and applied the (C++) closed itemset mining algorithm LCM (version 2 [17]). Whereas this algorithm only computes concept intents, it is known to be one of the most efficient for that task. To compute with pattern structures, we turned the many-valued context into a set of partitions over G (one for each attribute $m \in M$) and applied a slight (Java) modification of the algorithm CloseByOne [15]. Indeed, the latter can be easily adapted by changing the definition of both intersection and subsumption test, used for closures computation (a detailed explanation for another instance of pattern structures can be found in [14]). As such, this method computes pattern concepts, i.e. both pattern extent and intent.

Table 2 gives the details of the datasets and their derived formal context we experiment with. Note that in column $|\mathcal{B}_2(G)|$, formal objects (g,h) with empty description, i.e. $\{(g,h)\}' = \emptyset$ for any $g, h \in G$, are not taken into account. Table 3 gives the execution time of both methods. For pattern structures, execution times include the reading of the data, their process to a set of partitions and the CloseByOne execution. Concerning formal contexts, execution times include data reading and process with LCM, while the time to build the formal context is not taken into account. In both case, algorithms only output the number of

[5] `http://archive.ics.uci.edu/ml/datasets.html`

patterns. The experiments were carried out on an Intel Core i7 CPU 2.40 Ghz machine with 5 GB RAM.

	(G, M, W, I)		$(\mathcal{B}_2(G), MI)$		
Dataset	$\lvert G\rvert$	$\lvert M\rvert$	$\lvert\mathcal{B}_2(G)\rvert$	Avg. $\lvert g'\rvert$	Density
iris	150	5	4,363	1.38	27.56%
hepatitis	155	20	11,935	9.02	45.08%
glass	214	10	19,601	1.74	17.43%
imports-85	205	26	20,904	6.24	23.99%
balance-scale	625	5	143,236	1.67	33.35%
crx	690	16	236,633	5.53	43.54%
flare	1,066	13	567,645	8.79	67.60%
abalone	4,177	9	3,752,318	1.19	13.18%
krkopt-25%	7,013	7	20,115,505	1.84	26.26%
krkopt-50%	14,027	7	76,547,447	1.72	24.59%
krkopt-75%	21,040	7	171,199,419	1.66	24.22%
krkopt-100%	28,056	7	299,171,478	1.67	23.88%
adult-25%	8,140	15	33,124,730	6.32	42.14%
adult-50%	16,280	15	132,507,392	6.34	42.28%
adult-75%	24,320	15	295,709,848	6.34	42.20%
adult-100%	32,561	15	530,077,524	6.33	42.21%

Table 2. Datasets and their characteristics

From Table 3, it can be observed than computing with formal contexts is faster for the smallest datasets, even *abalone* that holds more than 3 millions of formal objects. However, with bigger datasets, from 20 to 530 millions of objects, partition pattern structures are the only able to compute the set of concepts. This holds for 7 numerical attributes already, and is bolder with 15. It is indeed already known that complexity of computing FD is highly related with the number of numerical attributes M.

As already suggested in [14,11] in different settings, the explanation is that when working with simple descriptions (i.e. vectors of bits), computing an intersection is more efficient than when working with more complex descriptions. Indeed, partitions are encoded in our algorithm as vectors of bitvectors (i.e. partitions) and both intersections or inclusion tests computation require to consider all pairs of sets between the two partitions in argument. Although we used optimizations avoiding an exhaustive computation between all pairs (by considering a lectic order on parts), those operations are more complex than standard intersections and inclusion tests between sets. However, we need to compute much less intersections, thus the following trade-off. Pattern structures perform better with larger datasets. Formal objects (numerical attributes) map to concise descriptions (partitions) whereas they map with the equivalence class of the same partitions in the case of formal contexts. Consequently, pattern structures are preferred to formal contexts when the number of possible pairs of objects that

	CloseByOne		LCMv2	
Dataset	Pattern concepts	Time (ms)	Concept intents	Time (ms)
iris	26	13	26	**4**
balance-scale	30	**30**	30	76
flare	4,096	**258**	4,096	650
glass	133	373	133	**41**
crx	9,528	4,367	9,528	**112**
abalone	252	5,887	252	**692**
hepatitis	95,576	11,178	95,576	**122**
imports85	205,623	228,877	205,623	**112**
krkopt-25%	126	**195**	126	5,441
krkopt-50%	126	**352**	N/A	N/A
krkopt-75%	126	**631**	N/A	N/A
krkopt-100%	126	**896**	N/A	N/A
adult-25%	10,881	**43,949**	N/A	N/A
adult-50%	12,398	**152,242**	N/A	N/A
adult-75%	13,133	**316,250**	N/A	N/A
adult-100%	13,356	**520,431**	N/A	N/A

Table 3. Comparing pattern structures and formal context representations. N/A means that the computation was intractable for memory issues.

agree for one or more attributes is high ($|\mathcal{B}_2(G)|$). Finally, let us recall that execution times for formal contexts do not include the scaling procedure time, since such procedure is highly dependent of I/O performances. We simply remark that conceptual scaling lengths more than 5 minutes for the dataset *adult-100%* resulting in a text-file of more than 10 giga-bytes (in the standard format of itemset mining algorithms: each line corresponds to an object described by the indexes of its attributes separated by a space).

To conclude, even with a simple Java implementation of CloseByOne (computing both extents and intents of pattern concepts), we gave here a proof of concept that pattern structures reveal themselves as a good trade off to overcome scaling difficulties, i.e. for computing the set of concepts whose lattice is equivalent to the one obtained after conceptual scaling.

6 Conclusions

We have presented a method to compute the characterization of a set of functional dependencies that hold in a set of many-valued data, based on formal concept analysis plus pattern structures. From this characterization, it is simply a matter of applying well-known algorithms to compute the minimal set of dependencies that imply the whose set (otherwise known as the Duquenne-Guigues basis). There was already methods to compute the characterization of those dependencies using FCA: One possibility is using conceptual scaling, which is the classical method to deal with many-valued data in FCA. This paper proposes to

use pattern structures, because they have already been used successfully to deal with many-valued data ([14]).

The empirical results compare an algorithm based on scaling versus another based on pattern structures, and show that scaling is faster for small datasets, whereas pattern structures perform better for large datasets, precisely where the scaling-based algorithm is not able to compute the output. This indicates that this new paradigm is more scalable in terms of time and memory. Since datasets tend to become larges and contain more attributes, this scalability may be a much important feature than speed in small datasets.

The results in this paper present a new paradigm for computing a characterization of functional dependencies that outperforms algorithms based on the *classical* conceptual scaling, which shows the interest of pattern structures for dealing with many-valued data within FCA. We think that the results that have been presented open the possibility to adapt this pattern structures based framework to other kinds of dependencies, namely, multi-valued dependencies and similar constraints that may be found in different fields.

7 Acknowledgements

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) that allowed professor Napoli to visit the Universitat Politècnica de Catalunya.

References

1. S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases, 1995. Addison-Wesley.
2. J. Baixeries and J. L. Balcázar. Discrete Deterministic Data Mining as Knowledge Compilation. Proceedings of Workshop on Discrete Mathematics and Data Mining in SIAM International Conference on Data Mining, 2003.
3. J. Baixeries. A Formal Concept Analysis framework to model functional dependencies. Mathematical Methods for Learning, 2004.
4. J. Baixeries. Lattice Characterization of Armstrong and Symmetric Dependencies. Ph. Thesis, 2007.
5. C. Beeri, R. Fagin,and J. H. Howard. A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations. Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data, Toronto, Canada, August 3-5, 1977.
6. N. Caspard and B. Monjardet. The Lattices of Closure Systems, Closure Operators, and Implicational Systems on a Finite Set: a Survey. Proceedings of the 1998 Conference on Ordinal and Symbolic Data Analysis (OSDA-98). Discrete Applied Mathematics, 2003.
7. A. Day. The Lattice Theory of Functional Dependencies and Normal Decompositions. International Journal of Algebra and Computation Vol. 2, No. 4 409-431. 1992.

8. J. Demetrovics, G. Hencsey, L. Libkin and I. Muchnik. Normal Form Relation Schemes: a New Characterization. Acta Cybernetica, 1992.
9. J. Demetrovics, L. Libkin and I. Muchnik. Functional Dependencies in Relational Databases: a Lattice Point of View. Discrete Applied Mathematics, 1992.
10. V. Duquenne and J.L. Guigues. Familles Minimales d'Implications Informatives Resultant d'un Tableau de Donées Binaires. Mathematics and Social Sciences, 1986.
11. B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In: Delugach, H., Stumme, G. (eds.) Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001). pp. 129–142. LNCS 2120, Springer (2001)
12. B. Ganter and R. Wille: Formal Concept Analysis. Springer, Berlin (1999)
13. G. Grätzer. General Lattice Theory. Academic Press, 1978.
14. M. Kaytoue, Kuznetsov and A. Napoli. Revisiting Numerical Pattern Mining with Formal Concept Analysis. IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia.
15. S. O. Kuznetsov and S. Obiedkov: Comparing performance of algorithms for generating concept lattices. Journal of Experimental & Theoretical Artificial Intelligence 14(2/3), 189–216 (2002)
16. D. Simovici and R. Tenney. Relational Database Systems. Academic Press, 1995.
17. T. Uno, M. Kiyomi, and H. Arimura:Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2004.
18. J. D. Ullman. Principles of Database Systems. Computer Science Press, 1982.